

# **Assembly ARM**

## Set di istruzioni

Le istruzioni dell'architettura ARM possono essere suddivise in tre insiemi:

- **Istruzioni aritmetico-logiche** implementano le operazioni matematiche. Queste possono essere di tipo **aritmetiche** (somma, differenza, prodotto), **logiche** (operazioni booleane) o **relazionali** (comparazioni tra due valori).
- **Istruzioni di Branch** cambiano il controllo del flusso delle istruzioni, modificando il contenuto del Program Counter (R15). Le istruzioni di branch sono necessarie per l'implementazione di istruzioni di condizione (if-then-else), cicli e chiamate di funzioni.
- **Istruzioni di Load/Store** muovono dati da (load) e verso (store) la memoria principale.

## Istruzioni aritmetico-logiche

In linguaggio assembly ARM le istruzioni aritmetico-logiche hanno una **sintassi** del tipo

opcode{S}{condition} dest, op1, op2

- Per **opcode** si intende l'istruzione che deve essere eseguita.
- Il campo **{S}** è un suffisso opzionale che, se specificato, carica nel registro di stato CPSR il risultato dell'operazione compresi i flag di stato.

## Esempi istruzioni aritmetico-logiche

ADD R0, R1, R2

Carico in R0 la somma tra il contenuto del registro R1 e del registro R2 (indirizzamento a registro).

ADD R0, R1, #16

Carico in R0 la somma tra il contenuto del registro R1 e il numero decimale 16 (indirizzamento immediato).

ADD R0, R1, #0xF

Carico in R0 la somma tra il contenuto del registro R1 e il numero esadecimale F (indirizzamento immediato).

## Esempi istruzioni aritmetico-logiche

**ADDLT** R0, R1, R2

Carico in R0 la somma tra il contenuto del registro R1 e del registro R2 solamente se la condizione indicata con il suffisso **LT** è verificata.

**ADDS** R0, R1, R2

Carico in R0 la somma tra il contenuto del registro R1 e del registro R2, inoltre carico lo stato del risultato nei flags NZCV del registro di stato CPSR.

Per esempio, se il risultato della somma va in overflow i flag C (carry) e V (overflow) verranno settati a 1.

# Principali istruzioni aritmetico-logiche

Descrizione	Opcode	Sintassi	Semantica
Addizione	ADD	ADD Rd, R1, R2/#imm	$R_d = R_1 + R_2/#imm$
Sottrazione	SUB	SUB Rd, R1, R2/#imm	$R_d = R_1 - R_2/#imm$
Moltiplicazione	MUL	MUL Rd, R1, R2/#imm	$R_d = R_1 \times R_2/#imm$
Carica nel registro	MOV	MOV Rd, R1/#imm	$R_d \leftarrow R_1/#imm$
Carica negato nel registro	MVN	MVN Rd, R1/#imm	$R_d \leftarrow -(R_1/#imm)$
AND logico	AND	AND Rd, R1, R2/#imm	$R_d = R_1 \wedge R_2/#imm$
OR logico	ORR	ORR Rd, R1, R2/#imm	$R_d = R_1 \vee R_2/#imm$
OR esclusivo	EOR	EOR Rd, R1, R2/#imm	$R_d = R_1 \oplus R_2/#imm$
AND con complemento del secondo operando	BIC	BIC Rd, R1, R2/#imm	$R_d = R_1 \wedge \neg R_2/#imm$
Shift logico sinistro	LSL	LSL Rd, R1, R2/#imm	$R_d = R_1 \ll R_2/#imm$
Shift logico destro	LSR	LSR Rd, R1, R2/#imm	$R_d = R_1 \gg R_2/#imm$
Rotazione destra	ROR	ROR Rd, R1, R2/#imm	$R_d = R_1 \circlearrowright R_2/#imm$
Confronto	CMP	CMP R1, R2	NZCV $\leftarrow R_1 - R_2$
Negato del confronto	CMN	CMN R1, R2	NZCV $\leftarrow R_1 + R_2$

## Istruzioni di branch

Le istruzioni di **branch** (o **jump**) sono utili per il controllo del flusso di esecuzione delle istruzioni.

Le principali istruzioni sono **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, causando così un salto nel flusso di esecuzione delle istruzioni. Per comodità le procedure vengono identificate univocamente con dei nomi detti **label** (o **etichetta**).

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.

## Istruzione di Load e Store

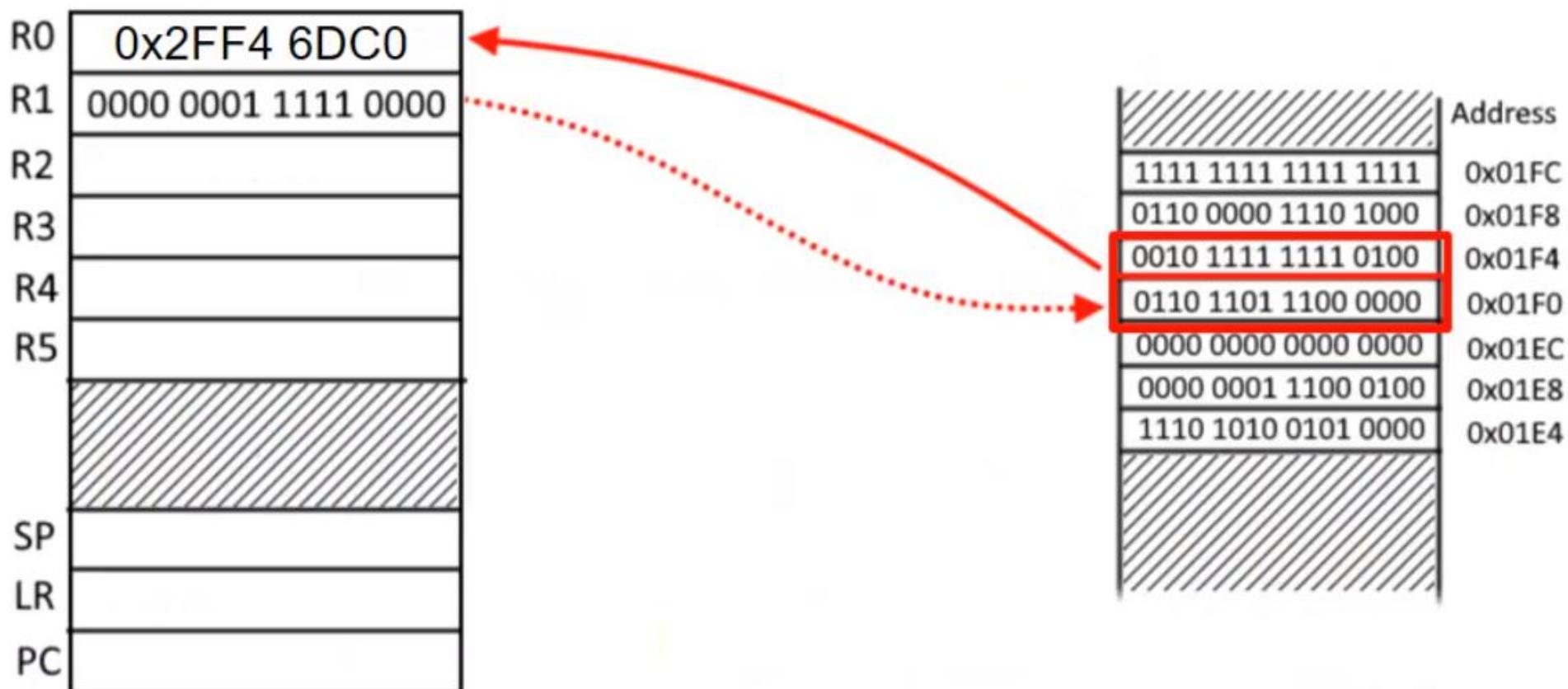
L'istruzione **LDR (Load Register)** carica in un registro destinatario un byte, una half word o una full word contenuta in una data locazione della memoria principale. La sintassi è

```
LDR{type}{condition} dest, [pointer]  
LDR{type}{condition} dest, [pointer,offset]
```

Il campo **dest** è il registro destinatario nel quale caricare il contenuto prelevato dalla memoria. Il campo **[pointer]** indica un **registro puntare** il quale definisce contenuto punta all'indirizzo in memoria della **cella** che si desidera referenziare. È possibile inoltre definire un **offset** il quale verrà sommato all puntatore.

LDR R0, [R1, #2]

Carico in R0 la **parola** in memoria puntata dal registro R1

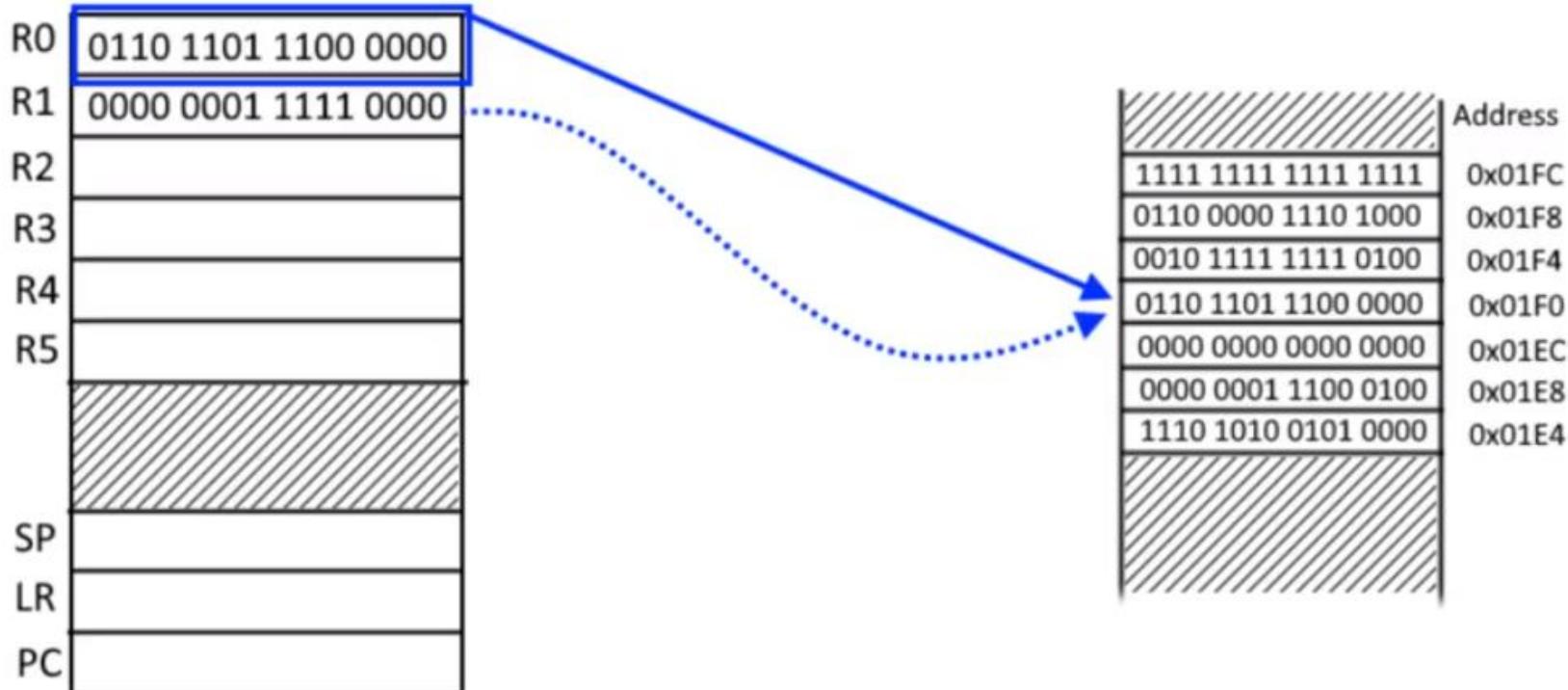


L'istruzione **STR** (**Store Register**) esegue esattamente l'operazione inversa di LDR; carica in memoria il contenuto di un registro **sorgente** all'indirizzo definito dal **puntatore**.

STR{condition} source, [pointer,offset]

STR R0, [R1]

Carico all'indirizzo puntato da R1 la parola contenuta nel registro R0



0x01F0 = 0000 0001 1111 0000

## Indirizzamento Immediato

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0			
0x00000081																																			
0x40000020	0	1																																	
0x10000008	0	0	0	1																															
0x04000002	0	0	0	0	0	1																												1	0
0x81000000	1	0	0	0	0	0	0	1																											
0x20400000			1	0	0	0	0	0	0	0	1																								
0x08100000				1	0	0	0	0	0	0	1																								
0x02040000					1	0	0	0	0	0	0	1																							
0x00810000						1	0	0	0	0	0	1																							
0x00204000							1	0	0	0	0	0	1																						
0x00081000								1	0	0	0	0	0	1																					
0x00020400									1	0	0	0	0	0	1																				
0x00008100										1	0	0	0	0	0	1																			
0x00002040											1	0	0	0	0	0	1																		

## Superare le difficoltà

MOV R0, #0xFFFFFFFF → MVN R0, #0  
R0 = NOT 0x00000000 = 0xFFFFFFFF

MOV R0, #0x55550000 → Errore a compile-time

## Superare le difficoltà

MOV R2, #0x55 // R2 = 0x00000055

ORR R2, R2, R2, LSL #8 // R2 = 0x00005555

LSL R2, R2,#16 // R2 = 0x55550000

oppure

LDR R2, =0x55550000 // R2 = 0x55550000

## Indirizzamento Immediato

31 30 29 28 ..... 15 14 13 12      11 10 9 8      7 6 5 4 3 2 1 0

	Posizione	Valore

0x00002040

.... 0010 0000 0100 0000



## Indirizzamento con shift immediato e con shift a registro

Se il bit 25 vale zero, nei bit 5 e 6 è scritto il tipo di shift da applicare tra quelli disponibili nel barrel shift (LSL, LSR, ASR, ROR e RRX). Mentre il bit 4 specifica il tipo di indirizzamento (immediato o con shift a registro).

Secondo Operando OP <sub>2</sub>				
Codifica Istruzione				Sintassi Istruzione
11..8	7	6 5 4	3..0	
valore		0 0 0	Rm	Rm, LSL #valore
Rs	0	0 0 1	Rm	Rm, LSL Rs
valore		0 1 0	Rm	Rm, LRR #valore
Rs	0	0 1 1	Rm	Rm, LSR Rs
valore		1 0 0	Rm	Rm, ASR #valore
Rs	0	1 0 1	Rm	Rm, ASR Rs
valore		1 1 0	Rm	Rm, ROR #valore
Rs	0	1 1 1	Rm	Rm, ROR Rs
0 0 0 0	0	1 1 0	Rm	Rm, RRX

## Indirizzamento con shift

// Indirizzamento con shift immediato

ADD R3, R3, R3, LSL #2 //  $R3 = R3 + R3 * 4$

RSB R2, R2, R2, LSL #4 //  $R2 = R2 * 16 - R2$

AND R0, R1, R2, RRX //  $R0 = R1 \text{ AND } RRX(R2)$

// Indirizzamento con shift a registro

// (5 bit LSB di R3)

ADD R0, R1, R2, LSL R3 //  $R0 = R1 + R2 * (2^R3)$

ORR R0, R0, R2, LSL R3 //  $R0 = R0 \text{ OR } R2 * (2^R3)$

AND R0, R0, R2, LSR R3 //  $R0 = R0 \text{ AND } R2 / (2^R3)$

## Istruzioni aritmetiche con saturazione

Se si utilizzano 32-bit per codificare i numeri interi con segno l'intervallo di rappresentazione si divide in due sottoinsiemi: da 1 a  $2^{31} - 1$  per i valori positivi e da  $-1$  a  $-2^{31}$  per quelli negativi.

Se sommando i numeri  $7 \cdot 2^{28}$  e  $2 \cdot 2^{28}$ , il risultato che si ottiene ( $8 \cdot 2^{28}$ ) non è positivo ma  $-2^{31}$

$$\begin{array}{r} 0x70000000 \\ + \\ 0x10000000 \\ \hline 0x80000000 \end{array}$$

Quando un valore supera un estremo, quest'ultimo è ricondotto al valore limite e si asserisce il flag Q nel registro di stato CPSR.

## Principali istruzioni aritmetiche con saturazione

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

Descrizione	MNEM	CODE	Semantica
Addizione con saturazione	<b>QADD</b>	10000	$Rd \leftarrow \text{sat}(Rm + Rn)$
Addizione con doppia saturazione	<b>QDADD</b>	10100	$Rd \leftarrow \text{sat}(Rm + \text{sat}(2 \cdot Rn))$
Sottrazione con saturazione	<b>QSUB</b>	10010	$Rd \leftarrow \text{sat}(Rm - Rn)$
Sottrazione con doppia saturazione	<b>QDSUB</b>	10110	$Rd \leftarrow \text{sat}(Rm - \text{sat}(2 \cdot Rn))$

$$\text{sat}(x) = \max(\min(x; 2^{31}-1); -2^{31})$$

## Istruzioni di Confronto

Le istruzioni di confronto utilizzano in ingresso due operandi ma non hanno un registro destinazione poiché il loro effetto è nella scrittura del registro di stato in base al valore risultato.

`<MNEM>{<PreCond>} <Rn>, OP2`

utilizzano lo stesso formato delle istruzioni aritmetiche/logiche → stesse regole per la modalità di indirizzamento del secondo operando.

Descrizione	MNEM	CODE	Semantica
Test bit	<b>TST</b>	1000	CPSR ← <Rn> · OP <sub>2</sub>
Test uguaglianza bit-a-bit	<b>TEQ</b>	1001	CPSR ← <Rn> ⊕ OP <sub>2</sub>
Comparazione	<b>CMP</b>	1010	CPSR ← <Rn> - OP <sub>2</sub>
Comparazione negativa	<b>CMN</b>	1011	CPSR ← <Rn> + OP <sub>2</sub>

## Istruzioni di Confronto

- **CMP** si comporta come **SUB**, **CMN** come **ADD**, perchè sottrarre un numero negativo equivale a sommarlo,
- **TST** come **AND**, **TEQ** come **EOR** poichè l'EXOR di numeri identici da risultato zero.

// Indirizzamento Immediato

CMP R1, #256 // Compara R1 con 256 e agg. flag

TST R2, #1 // Test bit LSB di R2 e agg. flag

// Indirizzamento a registro

TEQ R2, R3 // Test R2=R3? e agg. flag

// Indirizzamento con shift immediato

CMN R3, R3, LSL #4 // R3 + R3 \* 16 e agg. flag

TEQ R0, R1, RRX // R0 XOR R1 AND RRX(R1) e agg. flag

// Indirizzamento con shift a registro (5 bit LSB di R3)

TST R0, R1, LSL R3 // Test R0 = R1\*(2^R3) e agg. flag

CMN R0, R1, LSR R3 // R0 + R1/(2^R3) e agg. flag

## 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:

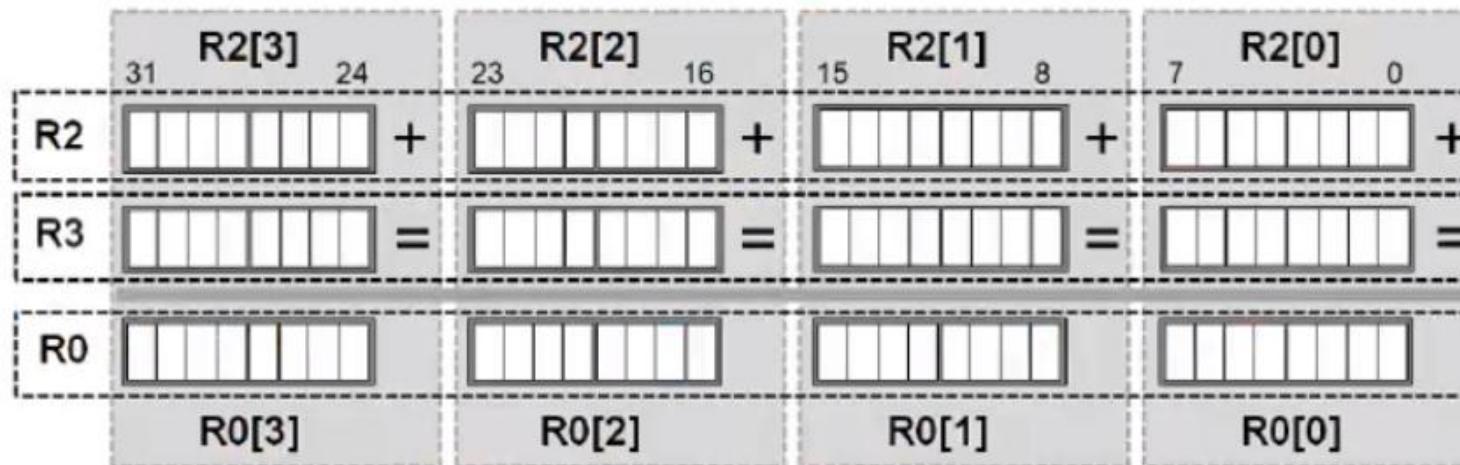
`<MNEM>{<PreCond>} <Rd>, <Rn>, <Rm>`

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



$$\begin{aligned}R0[0] &= R2[0] + R3[0] \\R0[1] &= R2[1] + R3[1] \\R0[2] &= R2[2] + R3[2] \\R0[3] &= R2[3] + R3[3]\end{aligned}$$

# Principali Istruzioni SIMD

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
	$Rm[0] \leftrightarrow Rm[1]$ ,	
<b>ADDSUBX</b>	$Rd[1] = Rn[1] + Rm[1]$ , $Rd[0] = Rn[0] - Rm[0]$	halfword
	$Rm[0] \leftrightarrow Rm[1]$ ,	
<b>SUBADDX</b>	$Rd[1] = Rn[1] - Rm[1]$ , $Rd[0] = Rn[0] + Rm[0]$	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	bit GPSR
S	con segno, modulo $2^8$ o $2^{16}$	GE[0÷3]
Q	con segno e con saturazione	
SH	con segno e risultati dimezzati	
U	senza segno, modulo $2^8$ o $2^{16}$	GE[0÷3]
UQ	senza segno e con saturazione	
UH	senza segno e risultati dimezzati	

## Istruzioni SIMD su array di byte

MNEM	Descrizione	Semantica
USAD8	Unsign. Sum Abs. Diff.	$\langle R_d \rangle \leftarrow \sum_0^3  R_m[i] - R_s[i] $
USADA8	Unsign. Sum Abs. Diff. Acc	$\langle R_d \rangle \leftarrow \langle R_n \rangle + \sum_0^3  R_m[i] - R_s[i] $

## Flag GE (Greater than or Equal)

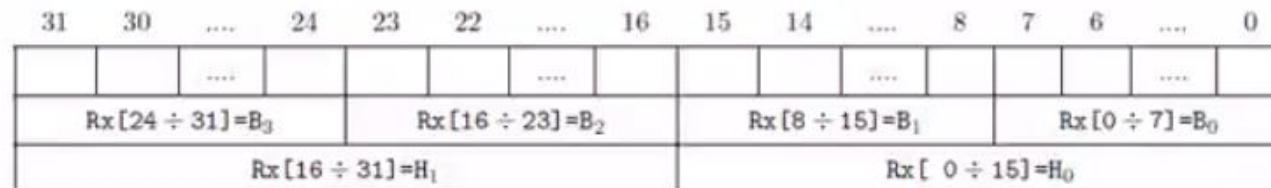
In ARMv6, le istruzioni SIMD impostano i flag GE della CPSR in base all'esito del risultato dei singoli byte o delle singole halfword che compongono la word.

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  $H_0$  è sinonimo di B (Bottom), talvolta indicato con l'acronimo LSH (Least Significant Halfword), mentre  $H_1$  è sinonimo di T (Top), qualche volta indicato con la sigla MSH (Most Significant Halfword).



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>

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

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

<MNEM>{<PreCond>} {<S>} <Rd>, <Rm>, <Rs>

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	$32 \times 16$	$Rd \leftarrow Rm \cdot Rs[y]$	$47 \div 16$
SMUAD	Sign. Mult, Add Dual	$16 \times 16$	$Rd \leftarrow Rm[B] \cdot Rs[B] + Rm[T] \cdot Rs[T]$	
SMUADX	Sign. Mult, Add Dual, eXch	$16 \times 16$	$Rd \leftarrow Rm[B] \cdot Rs[T] + Rm[T] \cdot Rs[B]$	
SMUSD	Sign. Mult, Sub Dual	$16 \times 16$	$Rd \leftarrow Rm[B] \cdot Rs[B] - Rm[T] \cdot Rs[T]$	
SMUSDX	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 moltiplicazione con tre operandi e risultato in word

<MNEM>{<PreCond>} {<S>} <Rd>, <Rm>, <Rn>, <Rs>

MNEM	Descrizione	Molt.	Semantica	Trunc.
MLA	Mult. Acc.	$32 \times 32$	$Rd \leftarrow Rn \cdot Rm \cdot Rs$	$31 \div 0$
SMLAxy	Sign. Mult. Long Acc	$16 \times 16$	$Rd \leftarrow Rn \cdot Rm[x] \cdot Rs[y]$	
SMLAWy	Sign. Mult. Word Acc	$32 \times 16$	$Rd \leftarrow Rn \cdot Rm \cdot Rs[y]$	$47 \div 16$
SMLAD	Sign. Mult. Add acc. Dual	$16 \times 16$	$Rd \leftarrow Rn \cdot Rm[B] \cdot Rs[B] + Rm[T] \cdot Rs[T]$	
SMUADX	Sign. Mult. Add acc. Dual. eXch	$16 \times 16$	$Rd \leftarrow Rn \cdot Rm[B] \cdot Rs[T] + Rm[T] \cdot Rs[B]$	
SMLSD	Sign. Mult. Sub acc. Dual	$16 \times 16$	$Rd \leftarrow Rm[B] \cdot Rs[B] - Rm[T] \cdot Rs[T]$	
SMLSDX	Sign. Mult. Sub acc. Dual. eXch	$16 \times 16$	$Rd \leftarrow Rn \cdot Rm[B] \cdot Rs[T] - Rm[T] \cdot Rs[B]$	
SMMLA	Sign. MSW Mult Acc. trunc.	$32 \times 32$	$Rd \leftarrow Rn \cdot Rm \cdot Rs$	$63 \div 32$
SMMLAR	Sign. MSW Mult Acc. Round.	$32 \times 32$	$Rd \leftarrow Rn \cdot Rm \cdot Rs$	$63 \div 32$
SMMLS	Sign. MSW Mult Sub. trunc	$32 \times 32$	$Rd \leftarrow Rn - Rm \cdot Rs$	$63 \div 32$
SMMLSR	Sign. MSW Mult Sub. Round	$32 \times 32$	$Rd \leftarrow Rn - Rm \cdot Rs$	$63 \div 32$

## 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:

$<\text{MNEM}>\{<\text{PreCond}>\}\{<\text{S}>\} <\text{Rd}>, \text{OP}_2$

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	✓	Rm	<i>Branch, eXch. Thumb</i>	$R15 \leftarrow Rm$ Se $Rm[0]=0$ imposta modo ARM, altrimenti Thumb ( $Rm[0]=1$ )
BXJ	✓	Rm	<i>Branch, eXch. Java</i>	$R15 \leftarrow Rm$ 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	✓	Rm	<i>Branch, Link, eXch. Thumb</i>	$R14 \leftarrow \text{indirizzo istruz. succ.}$ $R15 \leftarrow Rm[31 \div 1]$ Se $Rm[0]=0$ imposta modo ARM, altrimenti Thumb ( $Rm[0]=1$ )

## 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	[<Rn>]	$indirizzo \leftarrow Rn$
		[<Rn>, #{}{+ -}<offset <sub>12</sub> >]	$indirizzo \leftarrow Rn \pm offset_{12}$
	registro	[<Rn>, {}{+ -}<Rm>]	$indirizzo \leftarrow Rn \pm Rm$
	regis. scal.	[<Rn>, {}{+ -}<Rm>, <SHIFT>]	$indirizzo \leftarrow Rn \pm Rm \text{ } <\text{SHIFT}>$
Pre-indiciz.	immediato	[<Rn>, #{}{+ -}<offset <sub>12</sub> >] !	$indirizzo \leftarrow Rn \pm offset_{12}$ $Rn \leftarrow indirizzo$
	registro	[<Rn>, {}{+ -}<Rm>] !	$indirizzo \leftarrow Rn \pm Rm$ $Rn \leftarrow indirizzo$
	registro scalato	[<Rn>, {}{+ -}<Rm>, <SHIFT>] !	$indirizzo \leftarrow Rn \pm Rm \text{ } <\text{SHIFT}>$ $Rn \leftarrow indirizzo$
Post-indiciz.	immediato	[<Rn>], #{}{+ -}<offset <sub>12</sub> >	$indirizzo \leftarrow Rn$ $Rn \leftarrow Rn \pm offset_{12}$
	registro	[<Rn>], {}{+ -}<Rm>	$indirizzo \leftarrow Rn$ $Rn \leftarrow Rn \pm Rm$
	registro scalato	[<Rn>], {}{+ -}<Rm>, <SHIFT>	$indirizzo \leftarrow Rn$ $Rn \leftarrow Rn \pm Rm \text{ } <\text{SHIFT}>$

## Istruzioni di Load e Store: esempi

// Offset immediato	
LDR R0,[R1,#2]	// R0<-memoria[R1+2]
// Offset a registro	
STR R0,[R1,R2]	// R0->memoria[R1+R2]
// Offset a registro scalato	
LDR R0,[R1,R2,LSL #3]	// R0<-memoria[R1+R2*8]
// pre-indiciz. immediato	
STR R0,[R1,#2]!	// R0->memoria[R1+2] , R1=R1+2
// Pre-indiciz. a registro	
LDR R0,[R1,R2]!	// R0<-memoria[R1+R2] , R1=R1+R2
// Pre-indiciz. a registro scalato	
STR R0,[R1,R2,LSL #3]!	// R0->memoria[R1+R2*8] , R1=R1+R2*8
// Post-indiciz. immediato	
LDR R0,[R1],#2	// R0<-memoria[R1] , R1=R1+2
// Post-indiciz. a registro	
STR R0,[R1],R2	// R0->memoria[R1] , R1=R1+R2

## Istruzioni di Load e Store: esempi

// Post-indiciz. a reg. scalato

LDR R0,[R1],R2,LSL #3 // R0<-memoria[R1], R1=R1+R2\*8

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

LDREQB R2,[R5,#5] // Se EQ allora R2<-primo byte memoria[R5+#5]  
// azzerà altri 3 byte MSB di R2

// per invocare un sotto programma ROUTINE

STR R3, ROUTINE

MOV PC,R3 // R3<-indirizzo(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>8</sub> >]	<i>indirizzo</i> $\leftarrow Rn \pm offset_8$	B	Byte senza segno
	registro	[<Rn>, {}{+ -}Rm]	<i>indirizzo</i> $\leftarrow Rn \pm Rm$	SB	Byte con segno
Pre-indiciz.	immediato	[<Rn>, #{}{+ -}offset <sub>8</sub> ]!	<i>indirizzo</i> $\leftarrow Rn \pm offset_8$ $Rn \leftarrow indirizzo$	H	Halfword senza segno
	registro	[<Rn>, {}{+ -}Rm]!	<i>indirizzo</i> $\leftarrow Rn \pm Rm$ $Rn \leftarrow indirizzo$	SH	Halfword con segno
Post-indiciz.	immediato	[<Rn>], #{}{+ -}<offset <sub>8</sub> >	<i>indirizzo</i> $\leftarrow Rn$ $Rn \leftarrow Rn \pm offset_8$	<non indicato>	
	registro	[<Rn>], {}{+ -}Rm	<i>indirizzo</i> $\leftarrow Rn$ $Rn \leftarrow Rn \pm Rm$		

## Esempi di istruzioni di Load e Store su registri multipli modo 4 indirizzamento

LDM R7 ,{R1,R4,R5} // LDM sinonimo di LDMIA  
STMDB R3! ,{R4-R6,R11-R12}  
STMFD R13!,{R0-R10, LR}  
LDMFD R13!,{R1-R5 , PC}

## Load e Store con accesso esclusivo alla memoria

```
<LDREX>{<PreCond>} <Rd>,[<Rn>]  
<STREX>{<PreCond>} <Rd>,<Rm>,{<Rn>}
```

- per la sincronizzazione dei processi in un ambiente multiprocessore basato su memoria condivisa.
- sono atomiche ed evitano di bloccare le risorse di sistema durante le fasi di accesso alla memoria.
- necessitano di un costrutto di monitor per garantire l'accesso esclusivo alla memoria condivisa

## Istruzioni di cambiamento di stato

CPS{IE|ID} <int\_flag> {,#<modo>}

Sintassi		Current Program Status Register	
int_flag	bit	significato	
	a	A	Imprecise data Abort
	i	I	Interrupt Request
	f	F	Fast Interrupt Request

M <sub>4</sub> M <sub>3</sub> M <sub>2</sub> M <sub>1</sub> M <sub>0</sub>	Modalità	Registri accessibili
1 0 0 0 0	Utente	PC, R0 ÷ R14, CPSR
1 0 0 0 1	FIQ	PC, R0 ÷ R7, R8.FIQ ÷ R14.FIQ, CPSR, SPSR.FIQ
1 0 0 1 0	IRQ	PC, R0 ÷ R12, R13.IRQ, R14.IRQ, CPSR, SPSR.IRQ
1 0 0 1 1	Supervisor	PC, R0 ÷ R12, R13.SVC, R14.SVC, CPSR, SPSR.SVC
1 0 1 1 1	Abort	PC, R0 ÷ R12, R13.ABT, R14.ABT, CPSR, SPSR.ABT
1 1 0 1 1	Undefined	PC, R0 ÷ R12, R13.UND, R14.UND, CPSR, SPSR.UND
1 1 1 1 1	System	PC, R0 ÷ R14, CPSR

## Esempi di istruzioni di cambiamento di stato

CPSIE a,#17	// abilita i data Abort imprecisi e cambia il modo FIQ
CPSID if	// disabilita le interruzioni FIQ e IRQ
CPS #31	// imposta il modo di funzionamento a System

## Store Return State onto a stack

Grazie a questa istruzione è possibile predisporre lo stato di ritorno da un exception handler su uno stack diverso da quello utilizzato automaticamente dal processore

SRS{<Modo Agg>}{}{<PreCond>} SP{!}, #<Modo>

SRS{<Modo Agg>}{}{<PreCond>} #<Modo>{!}

equivalenti

scrive il bit di writeback

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

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

## 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
```

## Istruzioni verso i coprocessori: trasferimento dati modo 5 di indirizzamento

**LDC** (Load to Coprocessor) e **STC** (Store from Coprocessor).

```
<MNEM>{<PreCond>} {L} <Pd>, <CRd>, <Indirizzamento>  
<MNEM>2 {L} <Pd>, <CRd>, <Indirizzamento>
```

Esempio

CDP P1, 3, C3, C2, C1, 4 // Coprocessore P1 OPCODE\_1=3 OPCODE2=4

Modo	Sintassi	Semantica
Offset immediato	[<Rn>, #+/-<offset <sub>8</sub> > · 4]	<i>indirizzo</i> ← Rn ± offset <sub>8</sub> · 4
Pre-indicizzato immediato	[<Rn>, #+/-<offset <sub>8</sub> > · 4] !	Rn ← Rn ± offset <sub>8</sub> · 4 <i>inidirizzo</i> ← Rn
Post-indicizzato immediato	[<Rn>], #+/-<offset <sub>8</sub> > · 4	<i>indirizzo</i> ← Rn Rn ← Rn ± offset <sub>8</sub> · 4
Non indicizzato	[<Rn>], <option>	<i>indirizzo</i> ← Rn

## Istruzioni verso i coprocessori: trasferimento dati

**MCRR** (Move to Coprocessor from two ARM Registers),

**MRRC** (Move to two ARM registers from Coprocessor)

```
<MNEM>{<PreCond>} <Pd>, <OPCODE>, <Rd>, <Rn>, <CRm>
```

```
<MNEM>2 <Pd>, <OPCODE>, <Rd>, <Rn>, <CRm>
```



## Istruzioni verso i coprocessori: trasferimento dati

### Esempi

MCR P0, 4, R1, CR1, CR2, 1 // ARM->P0 OPCODE\_1=4 OPCODE\_2=1  
// R1-> CR1, CR2

MRC P1, 5, R2, CR0, CR1, 3 // ARM<-P1 OPCODE\_1=5 OPCODE\_2=3  
// R2<- CR0, CR1

MCRR P3, 6, R1, R2, CR0 // ARM->P3 OPCODE=6 R1, R2 -> CR0

MRRC P5, 3, R4, R3, CR2 // ARM<-P5 OPCODE=3 R3, R4 <- CR2

## Istruzioni verso i coprocessori: quadro di sintesi

Tipi di istruzioni	MNEM	Possibile comportamento
<i>Elaborazione Dati</i>	CPD	<i>Coprocessor Data Operations</i> $CRd \leftarrow OPCODE_1(CRn, OPCODE_2(CRm))$
<i>Trasferimento in Memoria</i>	LDC	<i>Load Coprocessor Register</i> $CRd \leftarrow memoria[indiriz(Rn)]$
	STC	<i>Store Coprocessor Register</i> $memoria[indiriz(Rn \leftarrow OPCODE_1(CRd))]$
<i>Trasferimento nei Registri</i>	MCR	<i>Move to Coprocessor from ARM Register</i> $CRn \leftarrow OPCODE_1(Rd, OPCODE_2(CRm))$
	MCRR	<i>Move to Coprocessor from two ARM Registers</i> $CRm \leftarrow OPCODE_1(Rd, Rn)$
	MRC	<i>Move to ARM Register from Coprocessor</i> $Rd \leftarrow OPCODE_1(CRn, OPCODE_2(CRm))$
	MRRC	<i>Move to two ARM Registers from Coprocessor</i> $Rd, Rn \leftarrow OPCODE_1(CRn)$

## Istruzioni per generare eccezioni

L'istruzione SWI (Software Interrupt), genera una eccezione SWI che porta il processore nello stato ARM e la modalità cambia in Supervisor. Quindi il registro CPSR viene salvato nel registro SPSR SVC

```
SWI{<PreCond>} <costante24>
```

## ISR

### **SWI\_Handler**

```
STMFD SP!,{R0-R12,LR}    // Salva i registri
LDR R0,[LR, #-4]          // calcola l'indirizzo dell'istruzione SWI chiamante
BIC R0,R0,#0xFF000000    // R0=costante in SWI ottenuta mascherando la Most
                           Significant Word
MOV R1, R0, LSR #8        // estraе l'offset della routine
ADR R2, TabellaRif        // estraе indirizzo di inizio della tabella dei riferimenti
LDR R15,[R2,R1,LSL #2]     // Salta alla corretta routine
LDMDF SP!,{R0-R12,LR}^    // Ripristina i registri
END                      // Fine dell'handler
```

## Istruzioni per la gestione dei dati: packing

Le istruzioni per il packing dei dati sono due:

- PKHBT (Pack Halfword Bottom Top)

PKHBT{<PreCond>} <Rd>, <Rn>, <Rm> {, LSL #<val>}

$$<\text{RdL}> \leftarrow <\text{RnL}>$$

$$<\text{RdH}> \leftarrow \begin{cases} <\text{RmH}> & \text{se LSL non è indicato} \\ \text{LSL}(\text{Rm}, \text{val}) [16 \div 31] & \text{altrimenti} \end{cases}$$

- PKHTB (Pack Halfword Top Bottom).

PKHTB{<PreCond>} <Rd>, <Rn>, <Rm> {, ASR #<val>}

$$<\text{RdL}> \leftarrow \begin{cases} <\text{RmL}> & \text{se ASR non è indicato} \\ \text{ASR}(\text{Rm}, \text{val}) [0 \div 15] & \text{altrimenti} \end{cases}$$

## Istruzioni per la gestione dei dati: Unpacking

- permettono di ruotare (0, 8, 16 o 24 bit) il contenuto di un registro, estendendo il segno al byte o alle halfword
- possono e accumulare il valore ottenuto, nei vari incrementi, in un registro destinazione.
- Tutte le operazioni di addizione aritmetica sono e  
ettuate in modulo  $2^{16}$  o  $2^{32}$  garantendo così l'assenza di riporti.

```
<MNEM> {<PreCond>} <Rd>, <Rm> {, ROR #<val>}
```

- Alle radici di istruzioni di unpacking si possono aggiungere i suffissi:

Suffisso	Descrizione estesa	Significato
S	<i>Sign extension</i>	Estensione del segno: 1 numeri negativi, 0 numeri positivi
U	<i>Unsigned/zero</i>	Estensione senza segno, cioè con zero

## Istruzioni per la gestione dei dati: Unpacking

Radice	Nome esteso e semantica
XTB	<i>eXTend Byte to word</i> Ruota il registro di 0, 8, 16 o 24 bit, estrae il byte $B_0$ e ne estende il segno a 32-bit
XTH	<i>eXTend Halfword to word</i> Ruota il registro di 0, 8, 16 o 24 bit, estrae la halfword $H_0$ e ne estende il segno a 32-bit
XTB16	<i>eXTend two Bytes to 16-bit</i> Ruota il registro di 0, 8, 16 o 24 bit, estrae i byte $B_0$ e $B_2$ ed estende il segno di entrambi a 16-bit
XTAB	<i>eXTend Byte to word, Add</i> Ruota il registro di 0, 8, 16 o 24 bit, estrae il byte $B_0$ , estende il segno a 32-bit e somma il risultato in un registro destinazione
XTAH	<i>eXTend Halfword to word, Add</i> Ruota il registro di 0, 8, 16 o 24 bit, estrae la halfword $H_0$ , estende il segno a 32-bit e somma il risultato in un registro destinazione
XTAB16	<i>eXTend two Bytes to 16-bit, Add</i> Ruota il registro di 0, 8, 16 o 24 bit, estrae i byte $B_0$ e $B_2$ , estende il segno di entrambi a 16-bit e somma le due halfword in un registro destinazione

## Esempi

MOV R9,#0x000000FE

SXTB R0, R9 // R0=0xFFFFFFFFE

SXTH R1, R9 // R1=0x000000FE

SXTB16 R2, R8 // R2=0x0000FFFE

LDR R8,=0xFFFF0FFF0

UXTB R3, R8 // R3=0x000000F0

UXTH R4, R8 // R4=0x0000FFF0

UXTB16 R5, R8 // R5=0x00F000F0

## Istruzioni di saturazione

Categoria	MNEM	Nome esteso e semantica
Saturazione con segno	SSAT	<i>Signed Saturate</i> $Rd = \begin{cases} -2^{N-1} & \text{se } OP_2 < -2^{N-1} \\ OP_2 & \text{se } -2^{N-1} \leq OP_2 \leq 2^{N-1} - 1 \\ +2^{N-1} - 1 & \text{se } OP_2 > 2^{N-1} - 1 \end{cases}$
Saturazione senza segno	USAT	<i>Unsigned Saturate</i> $Rd = \begin{cases} 0 & \text{se } OP_2 < 0 \\ OP_2 & \text{se } 0 \leq OP_2 \leq 2^N \\ 2^N - 1 & \text{se } OP_2 > 2^N - 1 \end{cases}$

## Istruzione di conteggio zeri finali

- fornisce quanti bit adiacenti sono zero a partire dal bit più significativo, spostandosi verso la cifra meno significativa

```
CLZ{<PreCond>}<Rd>, <Rm>
```

### Esempio

```
LDR R1,=0x0000FFFF // Costante non esprimibile con MOV  
MOV R3,#0x10000000  
CLZ R2,R1           // R2=0x00000010  
CLZ R4,R3           // R4=0x00000003
```

## Istruzioni di inversione di byte

- utili per convertire dati dal formato little-endian a quello big-endian e, viceversa, da big-endian al little-endian

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

## Istruzioni di inversione di byte

### *Esempio*

LDR R1,=0x0FE10547

REV R2, R1 // R2=0x4705E10F

REV16 R3, R1 // R3=0xE10F4705

REVSH R4, R1 // R4=0x00004705

## Pseudo-istruzioni

Il linguaggio assembly mette a disposizione del programmatore un insieme di pseudo-istruzioni di utilità, che saranno poi tradotte dall'assemblatore in istruzioni che il processore comprende.

Alcuni esempi sono:

- No Operation (NOP)
- Shifting e rotazione
- Copia registro

## No Operation

- NOP non esegue alcun compito
- può essere utilizzata come segnaposto nel corpo del programma da sostituire in seguito con istruzioni attive
- per invalidare una istruzione esistente (es. un branch) per scopi di debug
- può richiedere al processore ARM un ciclo di clock una volta tradotta,

L'assembler ARM potrebbe tradurre la NOP con l'istruzione:

MOV R0, R0 // modalità ARM

oppure

MOV R8, R8 // modalità Thumb

## Shifting e rotazione

RRX {<PreCond>} {<S>} <Rd>, <Rn>

Mentre per le istruzioni LSL, LSR, ASR, ROR le sintassi possibili sono:

<MNEM>{<PreCond>} {<S>} <Rd>, <Rn>, Rs

<MNEM>{<PreCond>} {<S>} <Rd>, <Rn>, #valore

Il registro Rs indica nel suo byte meno significativo il numero di cifre da scorrere.

Il #valore è un numero intero compreso tra 1 e 31

## Shifting e rotazione

MNEM	Descrizione	Semantica
LSL	<i>Shift Left</i>	$\langle Rd \rangle \leftarrow \langle Rn \rangle \ll \#valore$
LSR	<i>Shift Right</i>	$\langle Rd \rangle \leftarrow \langle Rn \rangle \gg \#valore$
ASR	<i>Shift Right with sign extend</i>	$\langle Rd \rangle \leftarrow \langle Rn \rangle \gg \#valore$
ROR	<i>Rotate Right</i>	$\langle Rd \rangle, C \leftarrow \langle Rn \rangle \gg \#valore$
RRX	<i>Rotate Right with eXtend</i>	$[\langle Rd \rangle   C] \leftarrow [\langle Rd \rangle   C] \gg 1$

## Shifting e rotazione

L'istruzione CPY è una pseudo-istruzione sinonimo della MOV ma, differentemente da quest'ultima, non ha la possibilità di impostare i flag del registro di stato (S) e neanche di eseguire lo shift sul secondo operando.

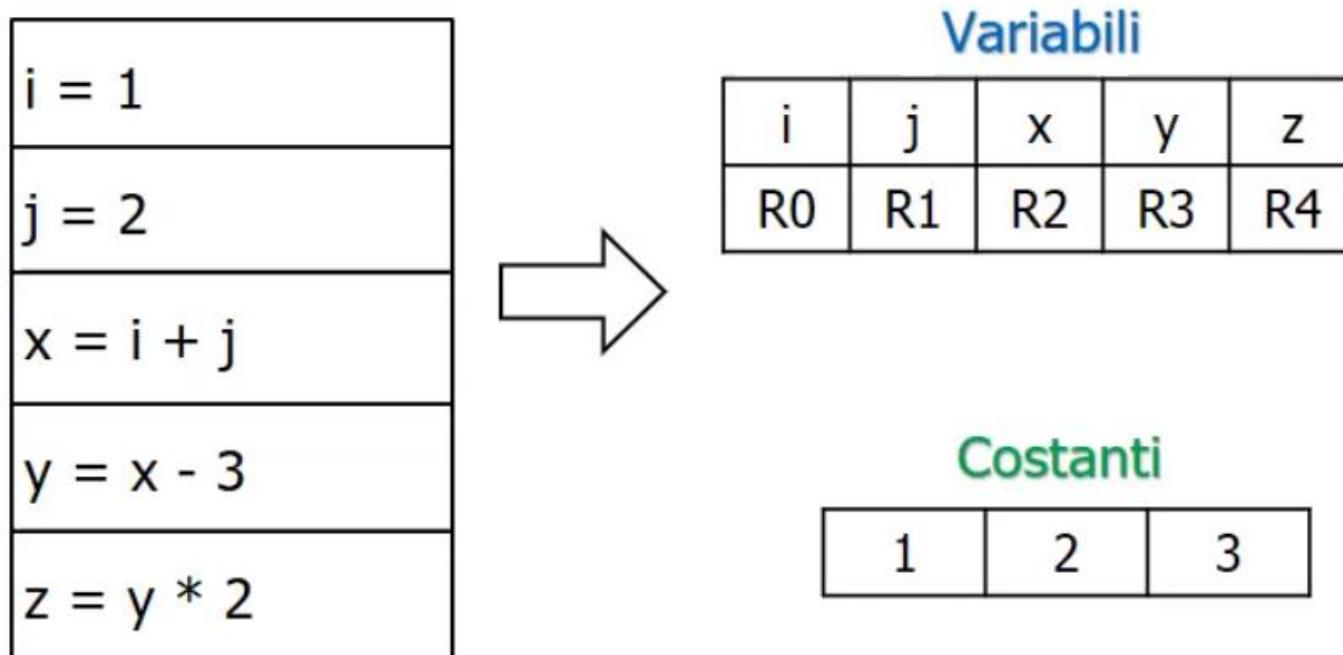
La sua sintassi risulta:

```
CPY{<PreCond>} <Rd>, <Rm>
```

# **Traduzione dei blocchi DNS in linguaggio Assembly ARM**

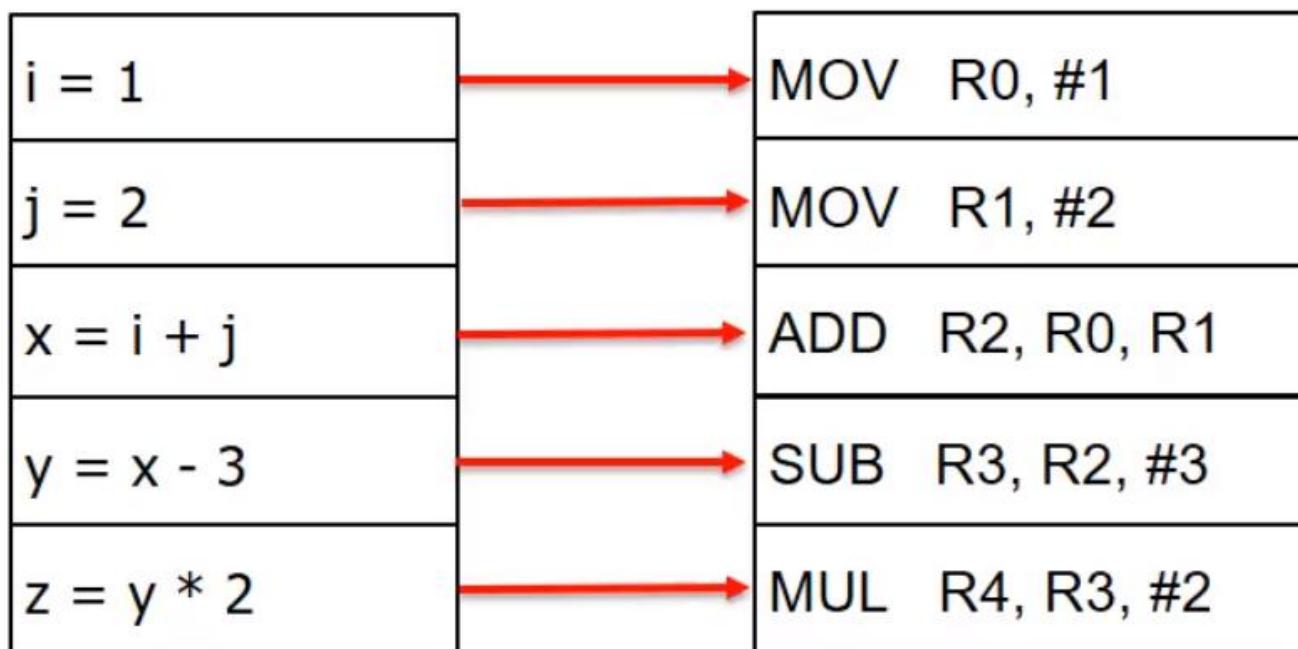
# Sequenza di blocchi

Inizialmente si cercano tutte le variabili e costanti che compaiono nel DNS. Si assegna ogni variabile ai registri disponibili, se il loro numero è tale da contenere tutte le variabili (ed il numero di byte tali da poter essere memorizzati in un registro), altrimenti occorre memorizzarle in memoria.

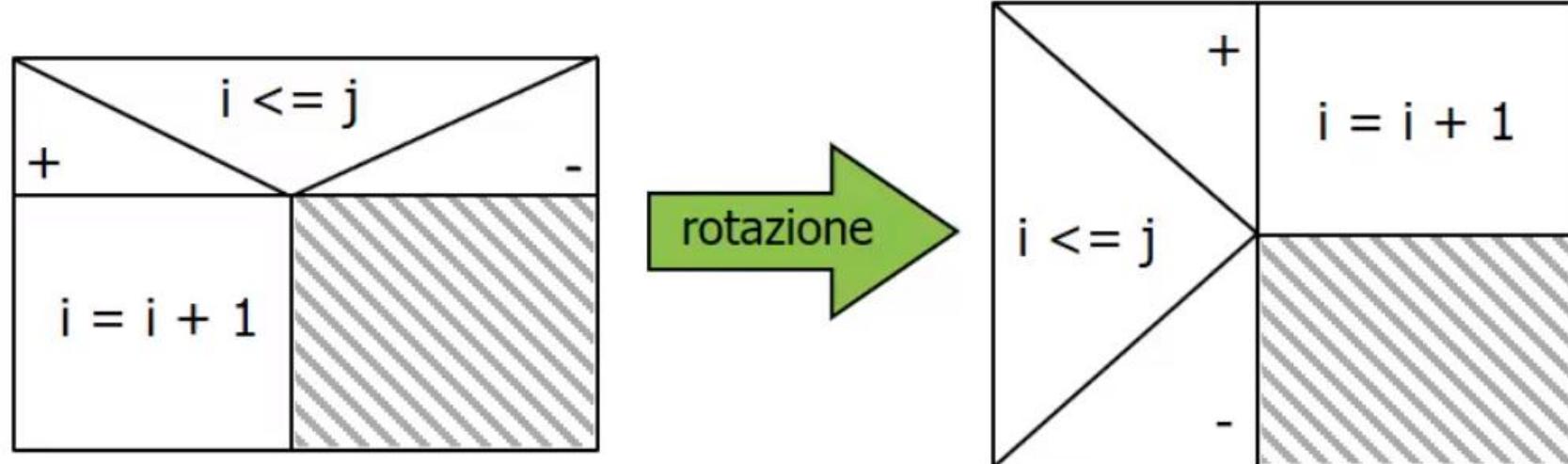


# Sequenza di blocchi

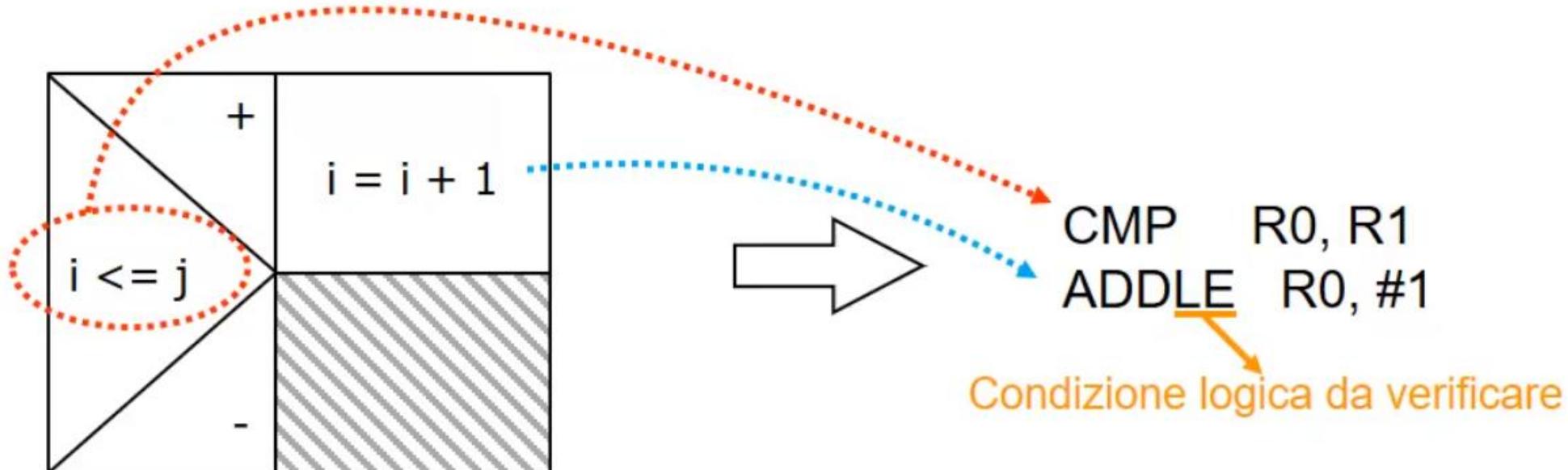
Se il blocco contiene una istruzione semplice, si traduce con la corrispondente istruzione assembly, altrimenti si segue la procedura che segue per ciascuna tipologia di blocco:



## Selezione singola



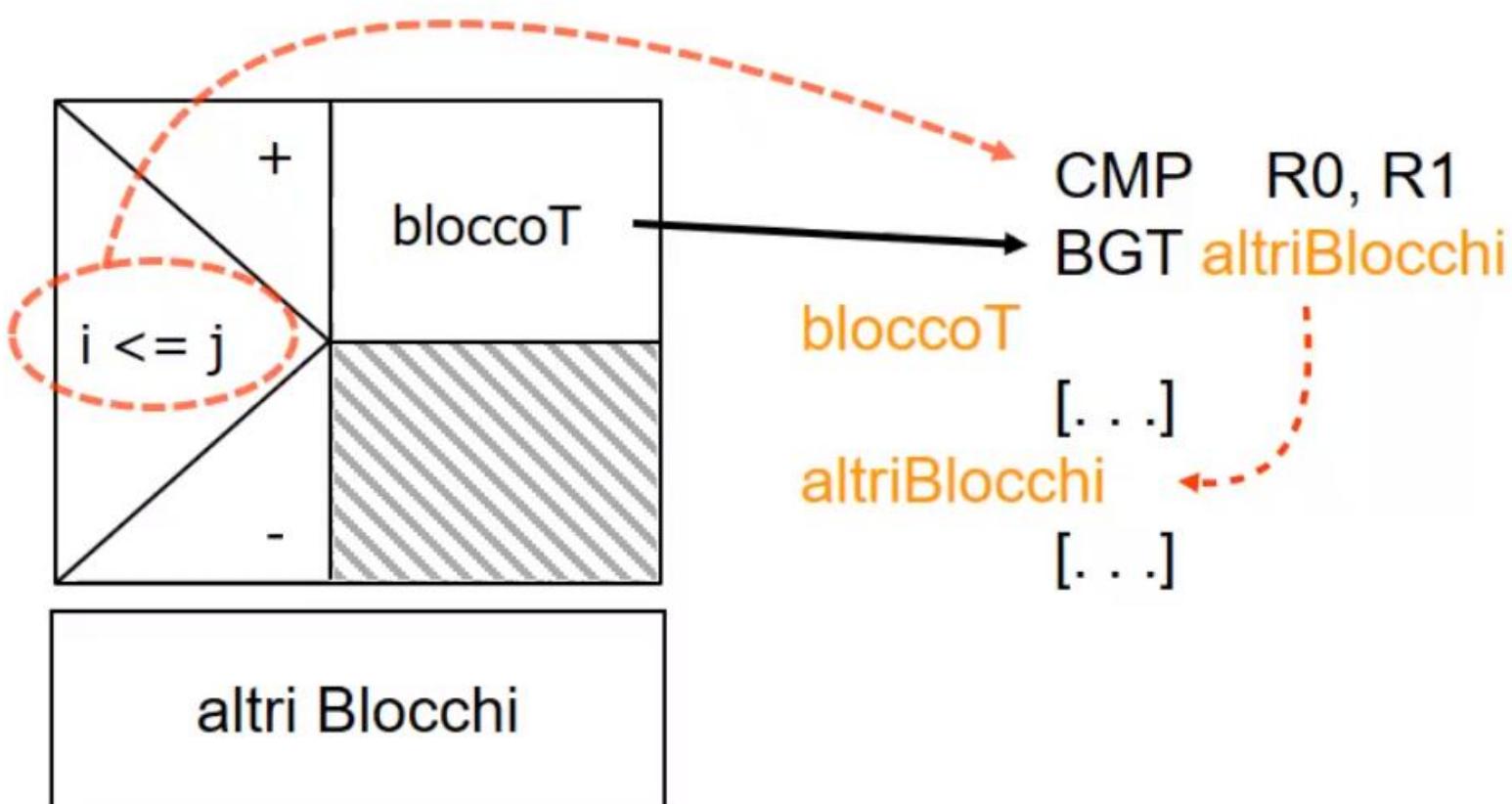
# Selezione singola



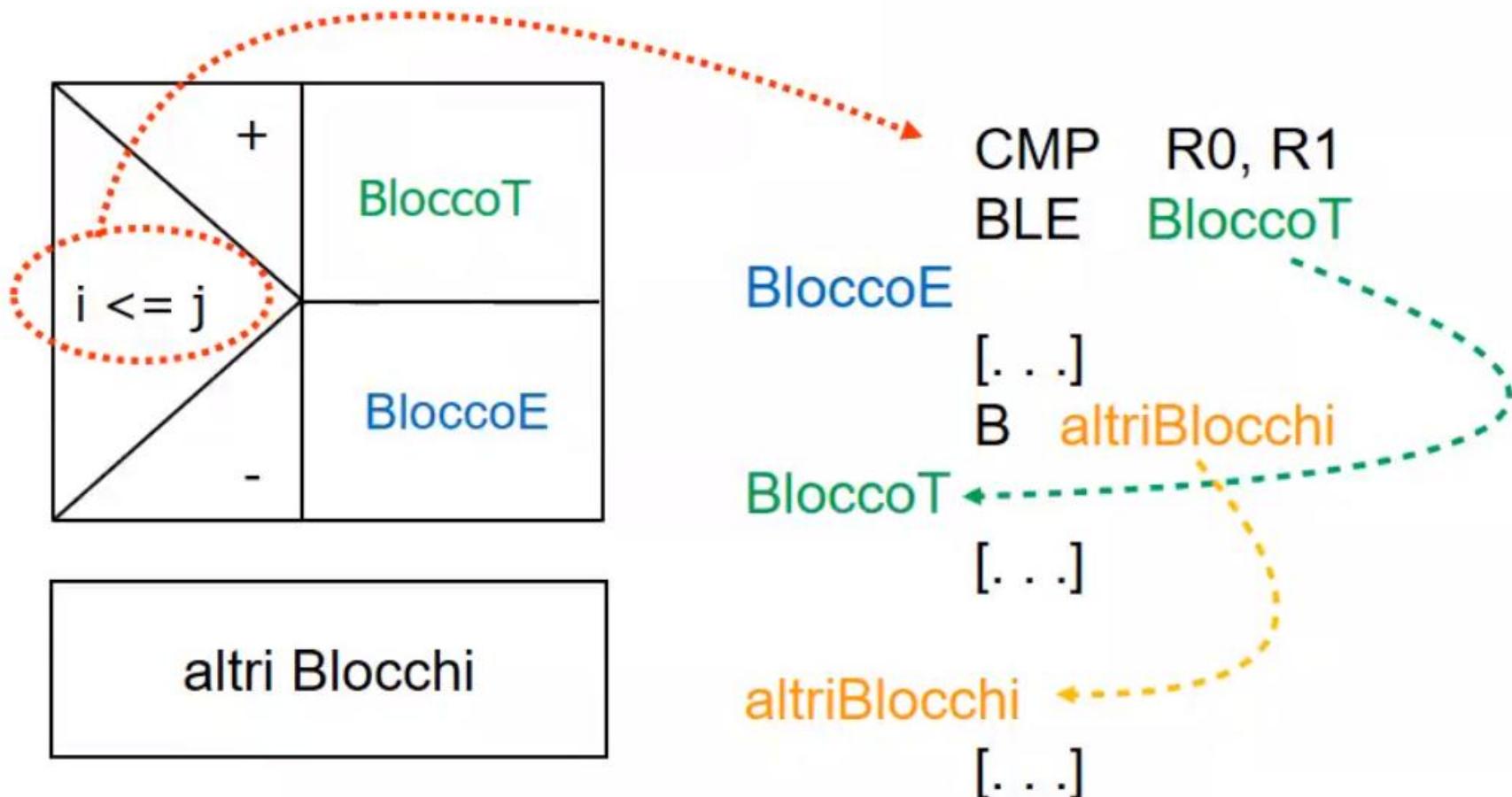
Se la condizione logica è verificata viene eseguita l'istruzione ADD, altrimenti il flusso di esecuzione proseguirà il normale corso.

# Selezione singola con branch

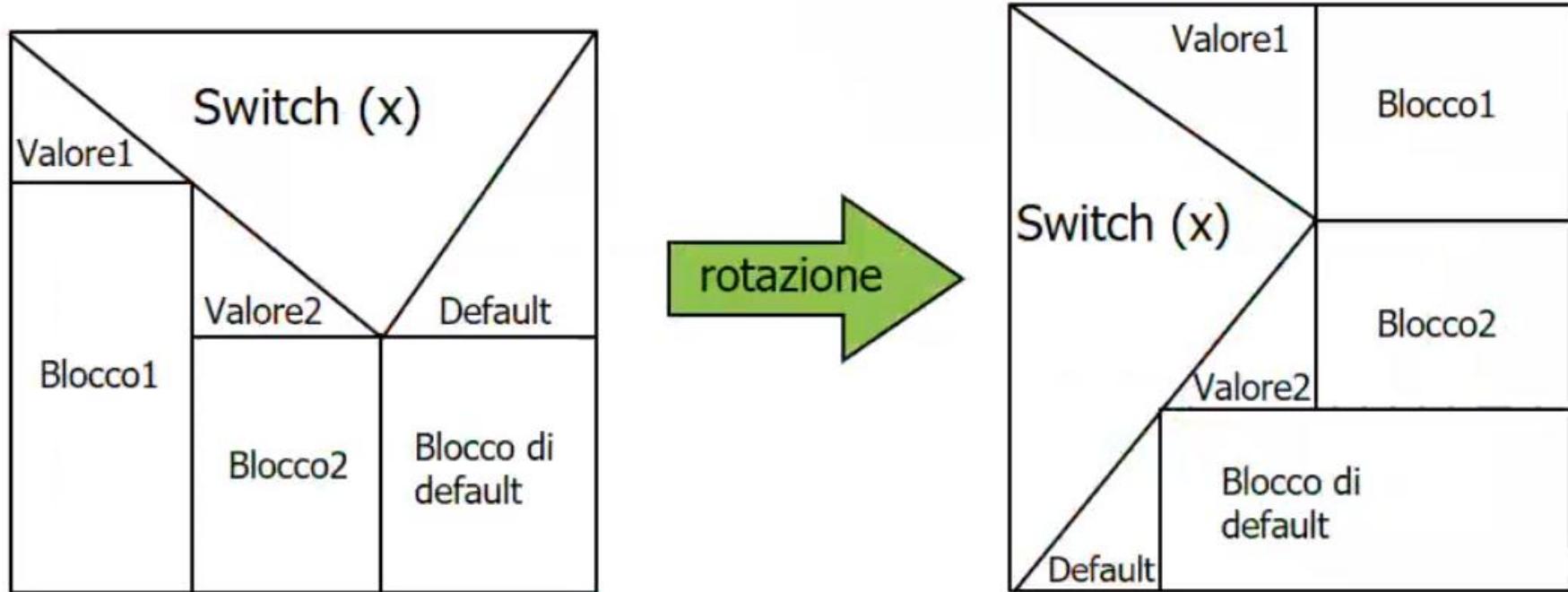
La selezione singola con esecuzione condizionata funziona nel caso ci sia una sola istruzione nel blocco, altrimenti è necessario utilizzare l'istruzione di branch.



## Selezione binaria



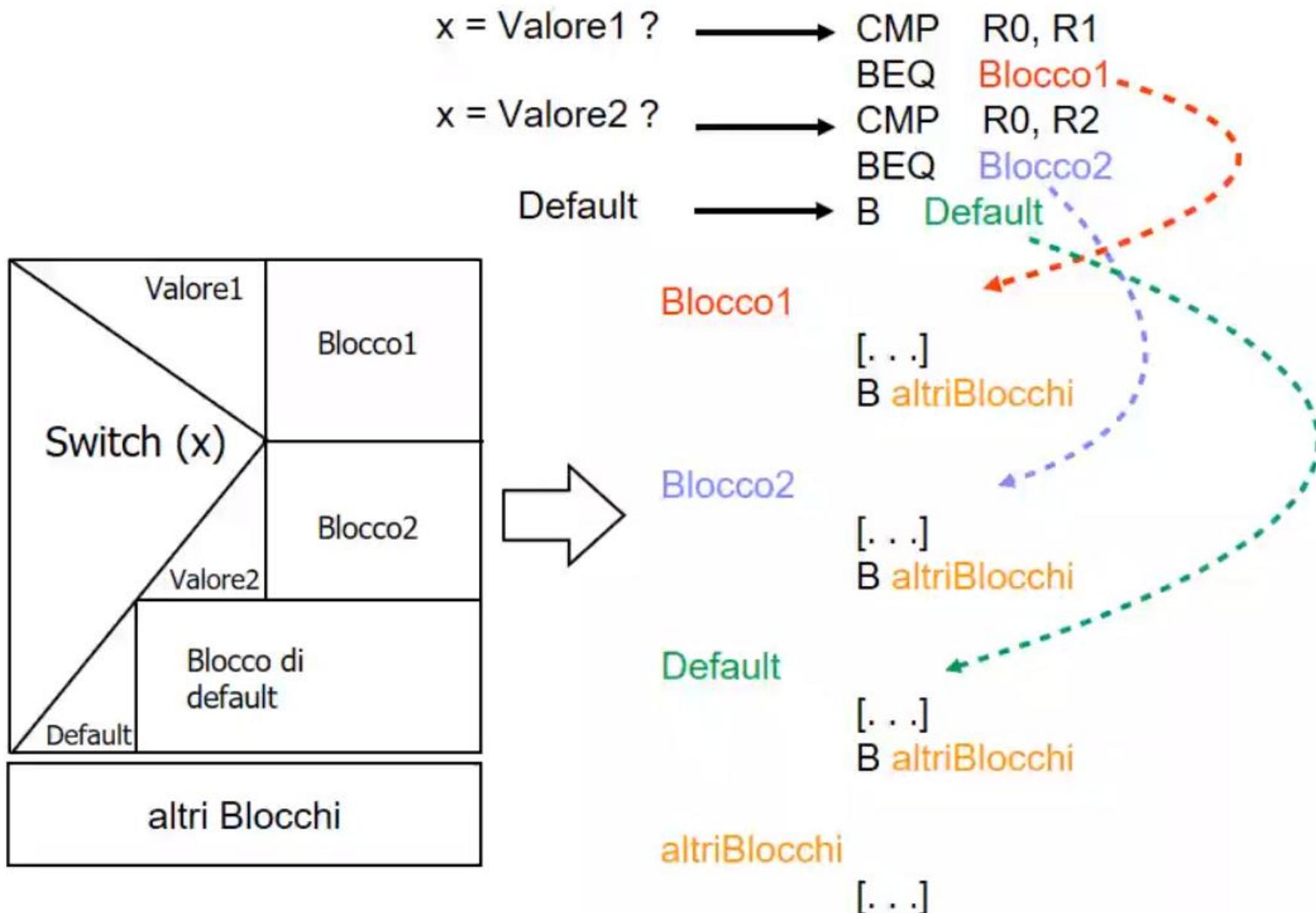
# Selezione multipla



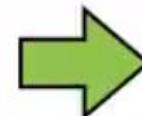
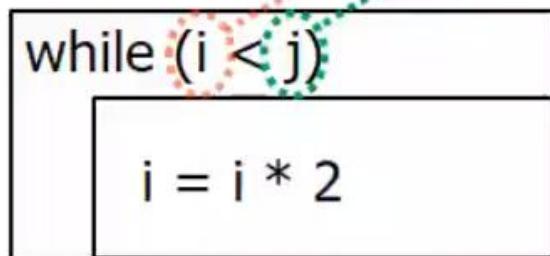
## Variabili

x	Valore1	Valore2
R0	R1	R2

## Selezione multipla

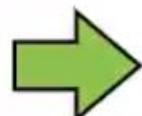
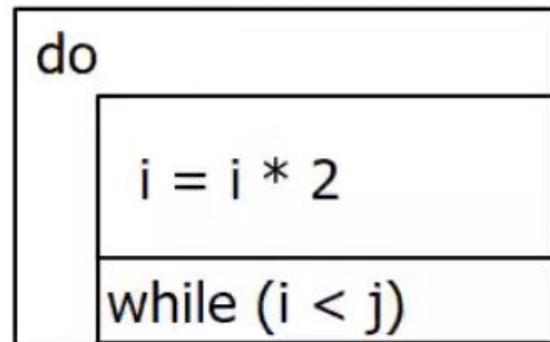


## Blocchi iterativi

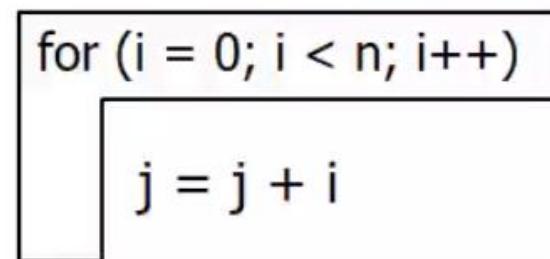


ciclo  
CMP R0, R1  
BGE end  
MUL R0, R0,#2  
B ciclo

end



ciclo  
MUL R0, R0,#2  
CMP R0, R1  
BLT ciclo



MOV R0, #0  
ciclo  
CMP R0, Rn  
BGE end  
ADD R1, R1, R0  
ADD R0, R0, #1  
B ciclo

end

## Massimo tra tre numeri interi

### *Esercizio*

Calcolare il massimo tra tre numeri interi  $x, y, z \in \mathbb{Z}$ .

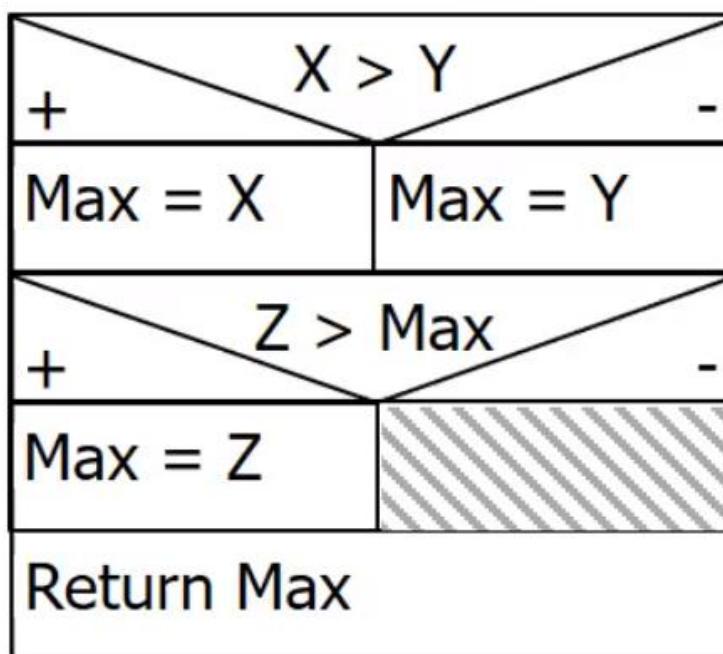
...proviamo a svolgerlo autonomamente, senza aiuto



## DNS

Il corrispondente diagramma DNS è:

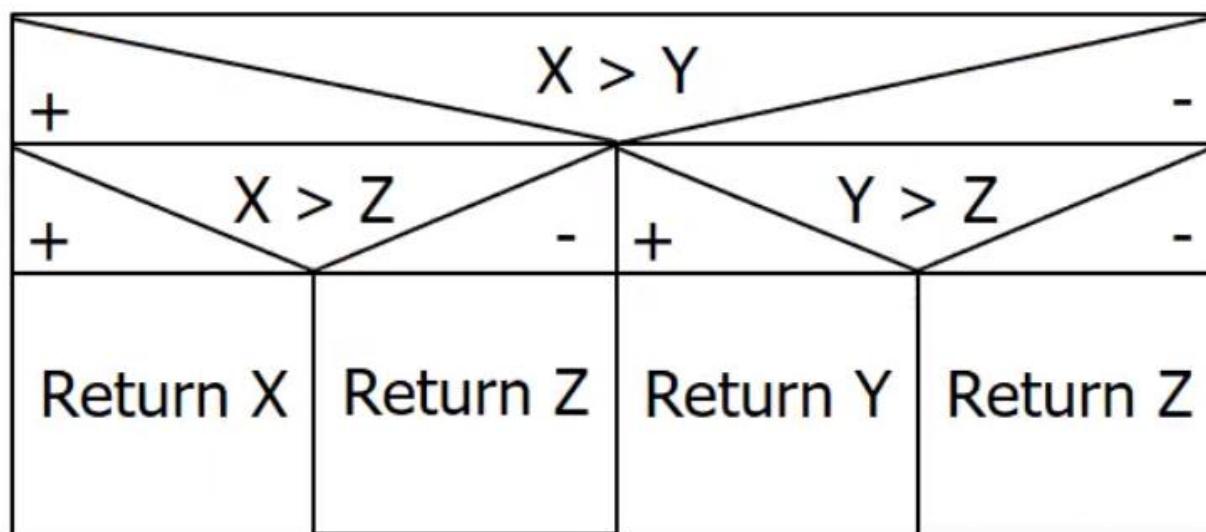
```
int max( X , Y , Z )
```



## Una soluzione alternativa

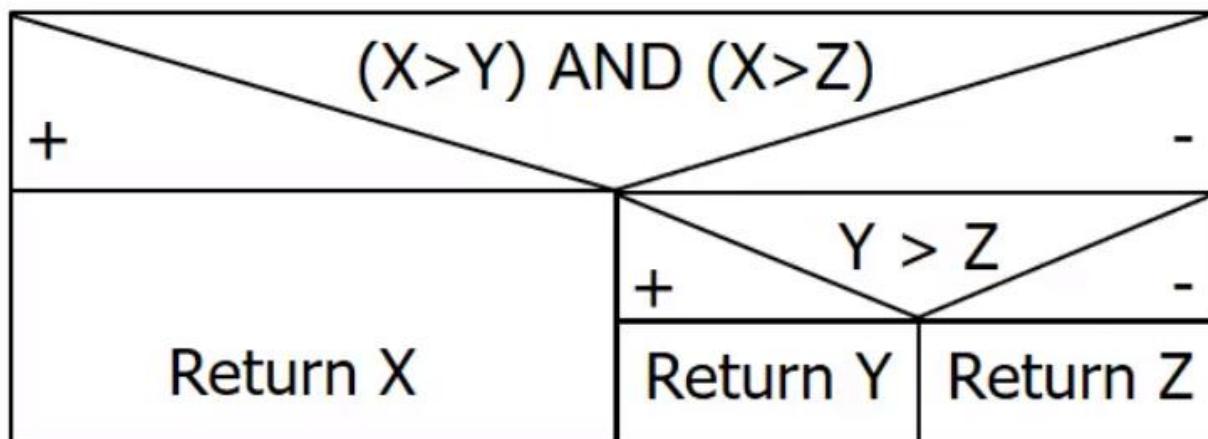
Tuttavia per ogni problema si possono trovare più soluzioni con diverse caratteristiche in termini di efficienza, complessità, riusabilità, ...

```
int max( X , Y , Z )
```



## Terza soluzione

```
int max( X , Y , Z )
```



## Codice assembly ARM finale

Sintassi GNU

```
.equ X, 10  
.equ Y, 30  
.equ Z, 20
```



Sintassi KEIL

```
X EQU 10  
Y EQU 30  
Z EQU 20
```

```
MOV R0, #X  
MOV R1, #Y  
MOV R2, #Z
```

```
CMP R0, R1 // X>Y ?  
MOVGT R3, R0 // MAX=X  
MOVLT R3, R1 // MAX=Y  
CMP R2, R3 // Z>MAX ?  
MOVGT R3, R2 // MAX=Z  
// R3=MAX
```

## Moltiplicazione

### *Esercizio*

Calcolare la moltiplicazione tra due numeri interi positivi  $x, y \in \mathbb{N}$  senza utilizzare l'istruzione MUL.

...proviamo a svolgerlo autonomamente, senza aiuto

