Assembly in ARM

Argomento:

- Struttura di un programma assembly
- Esempi di costrutti base
- Primi programmi

Struttura di un programma assembly ARM

Struttura programma assembly

- Un programma assembly consiste di uno o più file testuali (con estensione .s)
- Ogni file consiste di una sequenza di istruzioni assembly

Costruzione di un programma

- istruzione assembly ≠ istruzione macchina
 - Istruzione macchina è la rappresentazione binaria di un'istruzione assembly
 - (quasi) perfetta corrispondenza
- Il programma eseguibile viene costruito dal codice assembly tramite il **compilatore**
 - La compilazione consiste di due fasi: assemblaggio e linkaggio
- Il programma eseguibile contiene sia istruzioni macchina che dati
 - Contiene un punto di ingresso che rappresenta la prima istruzione del programma
- Esempio:
 - gcc -o sum sum.s → costruisce il programma eseguibile sum a partire dal sorgente assembly sum.s

Cosa manca per scrivere un programma?

Come sommiamo i primi 0x100 numeri?

• R0: contatore

• R1: somma parziale

```
mov r0, #0x100
mov r1, #0

add r1, r1, r0
subs r0, r0, #1
bne ????

str r1, [???]
```

Aiutare il compilatore

- Le istruzioni assembly non bastano per costruire un programma eseguibile
- Bisogna inoltre permettere di:
 - Indicare la prima istruzione
 - Allocazione di dati (e.g., allocare una cella di memoria con un valore numerico)
 - Indicare delle righe di codice (e.g., nei salti condizionati)
 - Inserire commenti

• ...

Direttive e simboli

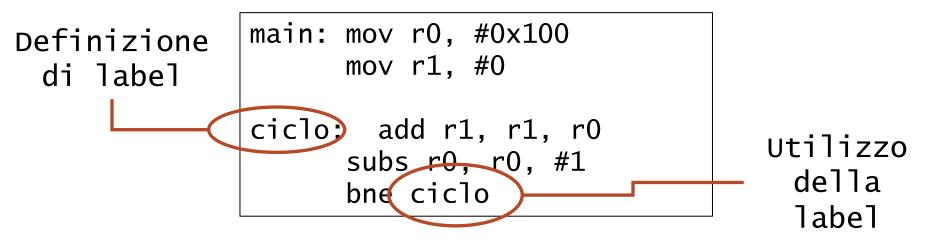
- Simboli: sono delle stringhe che denotano dei particolari valori (indirizzi in memoria, costanti numeriche,...)
 - Label: è un particolare simbolo utilizzato per indicare l'indirizzo in memoria di una particolare istruzione
- Direttive: sono istruzioni per l'assemblatore che permettono di
 - Definire simboli
 - Allocare aree della memoria per salvataggio di dati
 - Inizializzare il contenuto delle aree di memoria

Simboli e direttive scompaiono dopo la compilazione

Esempio di file assembly

```
global main
Simbolo
                  main:
                         ldr r0, =iterazioni
(label)
                         ldr r0, [r0] wumero iterazioni
                         mov r1, #0 @ Registro per spmme parziali
                  ciclo: add r1, r1, r0
                         subs r0, r0, #1
                                              Commento
                         bne ciclo
                         ldr r2, =output
                         str r1, [r2] @ salva soluzione in memoria
Istruzione
                  output: (.space 4
                  iterazioni: .word 0x100
  Architettura degli Elaboratori
                                              Direttiva
```

Definizione e uso di label



- Il compilatore calcola l'indirizzo dell'istruzione dove il label "ciclo" è definita
- Ogni volta che il label "ciclo" viene usato viene sostituita dall'indirizzo calcolato

Punto di inizio

 Ogni programma eseguibile deve avere un punto di inizio, definito dal label main

```
.global main
main: mov r0, #0x100
mov r1, #0
```

• E' necessario inserire prima della definizione del main la direttiva .global main

Commenti

Un commento inizia con @ e termina alla fine della riga

Esempio:

mov r0, #0x100 @ Numero iterazioni

Sintassi di un'istruzione assembly

• La forma generale delle linee di codice assembly è:

label: istruzione operandi @commento

- Ogni campo deve essere separato da uno o più spazi
- Tutti e tre le sezioni ...: ... @ ... sono opzionali
- I label devono iniziare dal primo carattere della riga
- Le istruzioni non cominciano mai dal primo carattere della riga: devono essere precedute da almeno 1 spazio
- L'assembly è case-insensitive

Allocazione di dati

- Le direttive permettono di allocare spazio per variabili e di inizializzare il loro valore
- Esempi:

maschera: .word 0xAAAAAAAA

stringa: .ascii "Pippo»

output: .space 4

- **.word, .hword, .byte** alloca in memoria e inizializza costanti (4/2/1 byte)
- .ascii alloca e inizializza una stringa
- .space, .skip allocano aree di memoria (.space inizializza a 0)

Segmenti di un programma

```
.text
.global main
main: ldr r0, =iterazioni
      ldr r0, [r0] @ Numero iterazioni
      mov r1, #0 @ Registro per somme parziali
ciclo: add r1, r1, r0
      subs r0, r0, #1
      bne ciclo
      ldr r2, =output
      str r1, [r2] @ salva soluzione in memoria
bss
output: .space 4
.data
iterazioni: .word 0x100
```

Segmenti di un programma assembly

Un programma assembly è diviso in tre segmenti:

- **TEXT** → segmento contenente codice (istruzioni) inizia con .text
- **DATA** → segmento contenente dati inizializzati inizia con .data
- **BSS** → segmento contenente dati non inizializzati inizia con .bss

Definizione di altri simboli

 Altre a label è possibile definire simboli per valore numerici con .equ

```
ciclo: add r1, r1, r0
subs r0, r0, #1
bne ciclo
```

```
.equ OFFSET, #1
ciclo: add r1, r1, r0
    subs r0, r0, #OFFSET
    bne ciclo
```

.equ non definisce un valore in memoria!

Esempi

.EQU cost ,5

Dichiara una costante cost=5

varA: .word 0xAA

Alloca una word con il valore 0xAA, l'etichetta varA contiene l'indirizzo in memoria della word

st_base: .skip 128

Alloca 128 byte, l'etichetta punta al byte con l'indirizzo più piccolo

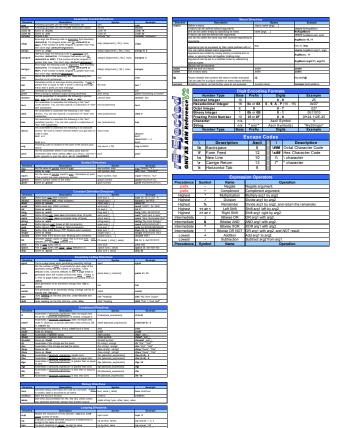
Quick references

In Moodle trovate le quick reference sul linguaggio ARM e sulle direttive

ARM® and Thumb®-2 Instruction Set Quick Reference Card

Key to Tables					
Rm {, <opsh>}</opsh>	See Table Register, optionally shifted by constant	<reglist></reglist>	A comma-separated list of registers, enclosed in braces { and }.		
<operand2></operand2>	See Table Flexible Operand 2. Shift and rotate are only available as part of Operand2.	<reglist-pc></reglist-pc>	As <reglist>, must not include the PC.</reglist>		
<fields></fields>	See Table PSR fields.	<reglist+pc></reglist+pc>	> As <reglist>, including the PC.</reglist>		
<psr></psr>	APSR (Application Program Status Register), CPSR (Current Processor Status Register), or SPSR (Saved Processor Status Register)	<flags></flags>	Either nzcvq (ALU flags PSR[31:27]) or g (SIMD GE flags PSR[19:16])		
C*, V*	Flag is unpredictable in Architecture v4 and earlier, unchanged in Architecture v5 and later.	§	See Table ARM architecture versions.		
<rs sh="" =""></rs>	Can be Rs or an immediate shift value. The values allowed for each shift type are the same as those	+/-	+ or (+ may be omitted.)		
	shown in Table Register, optionally shifted by constant.	<iflags></iflags>	Interrupt flags. One or more of a, i, f (abort, interrupt, fast interrupt).		
x,y	B meaning half-register [15:0], or T meaning [31:16].	<p_mode></p_mode>	See Table Processor Modes		
<imm8m></imm8m>	ARM: a 32-bit constant, formed by right-rotating an 8-bit value by an even number of bits.	SPm	SP for the processor mode specified by <p_mode></p_mode>		
	Thumb: a 32-bit constant, formed by left-shifting an 8-bit value by any number of bits, or a bit	<lsb></lsb>	Least significant bit of bitfield.		
	pattern of one of the forms 0xXYXYXYXY, 0x00XY00XY or 0xXY00XY00.	<width></width>	Width of bitfield. <width> + <lsb> must be <= 32.</lsb></width>		
<pre><prefix></prefix></pre>	See Table Prefixes for Parallel instructions	{X}	RsX is Rs rotated 16 bits if X present. Otherwise, RsX is Rs.		
{IA IB DA DB}	Increment After, Increment Before, Decrement After, or Decrement Before.	{!}	Updates base register after data transfer if ! present (pre-indexed).		
	IB and DA are not available in Thumb state. If omitted, defaults to IA.	{S}	Updates condition flags if S present.		
<size></size>	B, SB, H, or SH, meaning Byte, Signed Byte, Halfword, and Signed Halfword respectively.	{T}	User mode privilege if T present.		
	SB and SH are not available in STR instructions.	{R}	Rounds result to nearest if R present, otherwise truncates result.		

Operation		§	Assembler	S updates		s	Action		
Add	Add		ADD(S) Rd, Rn, <operand2></operand2>	N	Z	С	V	Rd := Rn + Operand2	N
	with carry		ADC(S) Rd, Rn, <operand2></operand2>	N	Z	C	V	Rd := Rn + Operand2 + Carry	N
	wide	T2	ADD Rd, Rn, # <imm12></imm12>					Rd := Rn + imm12, imm12 range 0-4095	T, P
	saturating {doubled}	5E	Q{D}ADD Rd, Rm, Rn					Rd := SAT(Rm + Rn) doubled: $Rd := SAT(Rm + SAT(Rn * 2))$	Q
Address	Form PC-relative address		ADR Rd, <label></label>					Rd := <label>, for <label> range from current instruction see Note L</label></label>	N, L
Subtract	Subtract		SUB{S} Rd, Rn, <operand2></operand2>	N	Z	С	V	Rd := Rn - Operand2	N
	with carry		SBC{S} Rd, Rn, <operand2></operand2>	N	Z	C	V	Rd := Rn - Operand2 - NOT(Carry)	N
	wide	T2	SUB Rd, Rn, # <imm12></imm12>					Rd := Rn - imm12, imm12 range 0-4095	T, P
	reverse subtract		RSB{S} Rd, Rn, <operand2></operand2>	N	Z	C	V	Rd := Operand2 - Rn	N
	reverse subtract with carry		RSC{S} Rd, Rn, <operand2></operand2>	N	Z	C	v	Rd := Operand2 - Rn - NOT(Carry)	A
	saturating {doubled}	5E	Q{D}SUB Rd, Rm, Rn					Rd := SAT(Rm - Rn) doubled: $Rd := SAT(Rm - SAT(Rn * 2))$	Q
	Exception return without stack		SUBS PC, LR, # <imm8></imm8>	N	Z	C	v	PC = LR - imm8, CPSR = SPSR(current mode), imm8 range 0-255.	
Parallel	Halfword-wise addition	6	<pre><pre><pre>fix>ADD16 Rd, Rn, Rm</pre></pre></pre>					Rd[31:16] := Rn[31:16] + Rm[31:16], Rd[15:0] := Rn[15:0] + Rm[15:0]	G
arithmetic	Halfword-wise subtraction	6	<pre><pre><pre>fix>SUB16 Rd, Rn, Rm</pre></pre></pre>					Rd[31:16] := Rn[31:16] - Rm[31:16], Rd[15:0] := Rn[15:0] - Rm[15:0]	G
	Byte-wise addition	6	<pre><pre><pre><pre>ADD8 Rd, Rn, Rm</pre></pre></pre></pre>					Rd[31:24] := Rn[31:24] + Rm[31:24], Rd[23:16] := Rn[23:16] + Rm[23:16], Rd[15:8] := Rn[15:8] + Rm[15:8], Rd[7:0] := Rn[7:0] + Rm[7:0]	G
	Byte-wise subtraction	6	<pre><pre><pre><pre>SUB8 Rd, Rn, Rm</pre></pre></pre></pre>					Rd[31:24] := Rn[31:24] - Rm[31:24], Rd[23:16] := Rn[23:16] - Rm[23:16], Rd[15:8] := Rn[15:8] - Rm[15:8], Rd[7:0] := Rn[7:0] - Rm[7:0]	G
	Halfword-wise exchange, add, subtract	6	<pre><pre><pre><pre>fix>ASX Rd, Rn, Rm</pre></pre></pre></pre>					Rd[31:16] := Rn[31:16] + Rm[15:0], Rd[15:0] := Rn[15:0] - Rm[31:16]	G
	Halfword-wise exchange, subtract, add	6	<pre><pre><pre>fix>SAX Rd, Rn, Rm</pre></pre></pre>					Rd[31:16] := Rn[31:16] - Rm[15:0], Rd[15:0] := Rn[15:0] + Rm[31:16]	G
	Unsigned sum of absolute differences	6	USAD8 Rd, Rm, Rs					Rd := Abs(Rm[31:24] - Rs[31:24]) + Abs(Rm[23:16] - Rs[23:16]) + $Abs(Rm[15:8] - Rs[15:8]) + Abs(Rm[7:0] - Rs[7:0])$	
	and accumulate	6	USADA8 Rd, Rm, Rs, Rn					Rd := Rn + Abs(Rm[31:24] - Rs[31:24]) + Abs(Rm[23:16] - Rs[23:16]) + Abs(Rm[15:8] - Rs[15:8]) + Abs(Rm[7:0] - Rs[7:0])	
Saturate	Signed saturate word, right shift	6	SSAT Rd, # <sat>, Rm{, ASR <sh>}</sh></sat>					Rd := SignedSat((Rm ASR sh), sat). <pre><pre><pre><pre>Rd := SignedSat((Rm ASR sh), sat). <pre><pre><pre><pre><pre><pre><pre><pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre></pre>	Q, R
	Signed saturate word, left shift	6	SSAT Rd, # <sat>, Rm{, LSL <sh>}</sh></sat>					Rd := SignedSat((Rm LSL sh), sat). <sat> range 1-32, <sh> range 0-31.</sh></sat>	Q
	Signed saturate two halfwords	6	SSAT16 Rd, # <sat>, Rm</sat>					Rd[31:16] := SignedSat(Rm[31:16], sat), Rd[15:0] := SignedSat(Rm[15:0], sat). <pre><pre><pre><pre></pre></pre><pre><pre></pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre><pre></pre></pre></pre>	Q
	Unsigned saturate word, right shift	6	USAT Rd, # <sat>, Rm{, ASR <sh>}</sh></sat>					Rd := UnsignedSat((Rm ASR sh), sat). <sat> range 0-31, <sh> range 1-31.</sh></sat>	Q, R
	Unsigned saturate word, left shift	6	USAT Rd, # <sat>, Rm{, LSL <sh>}</sh></sat>					Rd := UnsignedSat((Rm LSL sh), sat). <sat> range 0-31, <sh> range 0-31.</sh></sat>	Q
	Unsigned saturate two halfwords	6	USAT16 Rd, # <sat>, Rm</sat>					Rd[31:16] := UnsignedSat(Rm[31:16], sat), Rd[15:0] := UnsignedSat(Rm[15:0], sat). <pre><pre><pre><pre><pre><pre><pre><pre></pre></pre></pre></pre></pre></pre></pre></pre>	Q



Costrutti di base (if, for)

If-then-else

```
int a = 10;
int b = 20;
int c; // c= min(a,b)
if(a<b)</pre>
   c = a;
else
   c = b;
```

If-then-else in ARM

.data

```
addr_a: .word 10 @ inizializza a=10 addr_b: .word 20 @ inizializza b=20
```

.bss

addr_c: .skip 4 @ alloca c (4 byte)

.text

- - -

If-then-else in ARM (2)

.text LDR RO, =addr_a @ carica indirizzo di a LDR R1, [R0] @ copia a in R1 LDR RO, =addr_b @ carica indirizzo di b LDR R2, [R0] @ copia b in R2 LDR RO, =addr_c @ carica indirizzo di c CMP R1, R2 @ confronta R1(=a) e R2(=b) BGE else_case @ se R1>=R2 salta STR R1, [R0] @ imposta c=a B end_if else_case: STR R2, [R0] @ imposta c=b end_if: NOP

If-then-else in ARM (short version)

- - -

```
LDR RO, =addr_a @ carica indirizzo di a LDR R1, [R0] @ copia a in R1 LDR RO, =addr_b @ carica indirizzo di b LDR R2, [R0] @ copia b in R2 LDR R0, =addr_c @ carica indirizzo di c CMP R1, R2 @ confronta R1 e R2 STRLT R1, [R0] @ imposta c=a se a<b STRGE R2, [R0] @ imposta c=b se a>=b
```

For loop

```
int sum = 0;
int n = 100;
for(int i=1; i<=n; i++){
    sum = sum + i;
}</pre>
```

For loop in ARM

.data

addr_n: .word 100 @ inizializza n=100

.bss

add_sum: .skip 4 @ alloca sum (4 byte)

.text

. . .

For loop in ARM (2)

. . .

```
.text
```

```
LDR R3, = addr_n @ copia indirizzo di n

LDR R2, [R3] @ copia n in R2

MOV R1, #0 @ R1 contiene somma parziale

MOV R0, #0 @ R0 è il contatore i
```

- - -

For loop in ARM (3)

```
for_loop: ADD R1, R1, R0 @ calcola sum=sum+i
ADD R0, R0, #1@ incremento contatore
CMP R0, R2 @ confronta se i<=n
BLE for_loop @ se si, ripeti loop

LDR R3, =addr_sum
STR R1, [R3]
```