

ARM: le istruzioni assembly

03.b



C. Fantozzi, A. Gardich
(revisione di S. Congiu)



DEPARTMENT OF
INFORMATION
ENGINEERING
UNIVERSITY OF PADOVA



Struttura programma assembly

La forma generale delle linee di codice assembly è:

label : <sp> **istruzione** <sp> **operandi** <sp> **@commento**

1
U
III
52

Ogni campo deve essere separato da uno o più spazi

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

Tutti e tre le sezioni ... : ... @ ... sono opzionali

Esistono inoltre:

- direttive per l'assemblatore
- pseudo-istruzioni

Direttive per l'assemblatore

2

U
III
52

direttive: sono comandi per l'assemblatore, non per il processore

- **.text** specifica la sezione contenente il codice eseguibile
- **.data** specifica la sezione contenente dati inizializzati
- **.bss** specifica la sezione contenente dati non inizializzati

- **.end** specifica la fine del modulo sorgente
- **.global** definisce simboli visibili al di fuori del modulo stesso: utile per collegare più moduli (linking)

- **.long .short .byte** definiscono costanti (es. in **.data**)
- **.ascii** definisce una stringa
- **.lcomm .comm .skip** definiscono aree di mem. non inizializzate (locali al modulo, o visibili da altri moduli)

Esempio di file **.s**

3

U
III
52

```
Label  _start:
      .text
      .global _start
      mov r0, #0x100    @ Inizia il programma
      ciclo:
      subs r0, r0, #1
      bne ciclo

Istruzione  ldr pc, =0x112C @ esce al sistema

      .data
maschera:  .long 0xAAAAAAAA
stringa:   .ascii "Pippo"

      .bss
      .lcomm spazio, 100

      .end
```

Direttiva

Sezione

Classi di indirizzamento

4

U
I
I
52

Modo 1: per istruzioni di **elaborazione dati**

- ADC, ADD, AND, BIC, CMN, CMP, EOR, MOV, MVN, ORR, RSB, RSC, SBC, SUB, TEQ, TST

Modo 2: per **Load&Store di word o unsigned byte**

- LDR, LDRB, STR, STRB

Modo 3: per **L&S di halfword o signed byte**

- LDRH, LDRSB, LDRSH, STRH

Modo 4: per **L&S di registri multipli**

- LDMxx, STMxx

Modo 1 (indirizz. per elab. dati)

5

U
I
I
52

sintassi:

<istruzione3op> Rd, Rn, **<shifter_operand>**

<istruzione2op> Rd, **<shifter_operand>**

<shifter_operand> può essere:

- un valore immediato **#<valore>**
- un registro **Rm**
- un registro, dopo scorrimento specificato con:
 - un valore immediato **Rm, <sop> #<shift_imm>**
 - un registro **Rm, <sop> Rs**
- gli operatori **<sop>** disponibili sono:
ASR, LSL, LSR, ROR, RRX

Modo 1: esempi

<code>mov R0, #0</code>	@ R0 ← 0
<code>add R3, R3, #1</code>	@ R3 ← R3+1
<code>cmp R7, #1000</code>	@ cc ← (R7-1000)
<code>bic R9, R8, #0xff00</code>	@ R9 ← R8 and not 0xff00
<code>mov R2, R0</code>	@ R2 ← R0
<code>add R4, R3, R2</code>	@ R4 ← R3+R2
<code>mov R2, R0, LSL #2</code>	@ R2 ← R0*4
<code>add R9, R5, R5, LSL #3</code>	@ R9 ← R5+R5*8 = R5*9
<code>sub R9, R5, R5, ASR #3</code>	@ R9 ← R5-R5/8
<code>rsb R9, R5, R5, ASR #3</code>	@ R9 ← R5/8-R5
<code>mov R5, R3, RRX</code>	@ R5 ← R3 ruotato esteso a destra di una posiz.
<code>mov R7, R4, ROR R3</code>	@ R7 ← R4 ruotato a destra di R3 posizioni

6

U
III
52

Modo 2 (indirizz. per Word o uns. Byte)

sintassi:

`LDR|STR{B} Rd, <addressing_mode2>`

`<addressing_mode2>` è un indirizzamento indiretto con registro [Rn], che può avere una delle seguenti forme:

- offset immediato
- offset da registro
- offset da registro scalato
- pre-incremento immediato
- pre-incremento da registro
- pre-incremento da registro scalato
- post-incremento immediato
- post-incremento da registro
- post-incremento da registro scalato

7

U
III
52

Modo 2: offset

8

U
U
U
52

Offset immediato

- $[Rn, \#_{\pm} \langle \text{offset}_{12} \rangle]$
@ Rd \leftrightarrow $M[Rn \pm \langle \text{offset}_{12} \rangle]$

Offset da registro

- $[Rn, \pm Rm]$
@ Rd \leftrightarrow $M[Rn \pm Rm]$

Offset da registro scalato

- $[Rn, \pm Rm, \langle \text{sop} \rangle \# \langle \text{shift}_{\text{imm}} \rangle]$
@ Rd \leftrightarrow $M[Rn \pm (Rm \langle \text{sop} \rangle \# \langle \text{shift}_{\text{imm}} \rangle)]$

Modo 2: pre-incremento

9

U
U
U
52

Pre-incremento immediato

- $[Rn, \#_{\pm} \langle \text{offset}_{12} \rangle]!$
@ $Rn \leftarrow Rn \pm \langle \text{offset}_{12} \rangle$
@ Rd \leftrightarrow $M[Rn]$

Pre-incremento da registro

- $[Rn, \pm Rm]!$
@ $Rn \leftarrow Rn \pm Rm$
@ Rd \leftrightarrow $M[Rn]$

Pre-incremento da registro scalato

- $[Rn, \pm Rm, \langle \text{sop} \rangle \# \langle \text{shift}_{\text{imm}} \rangle]!$
@ $Rn \leftarrow Rn \pm (Rm \langle \text{sop} \rangle \# \langle \text{shift}_{\text{imm}} \rangle)$
@ Rd \leftrightarrow $M[Rn]$

Modo 2: post-incremento

10

U
III
52

Post-incremento immediato

- **[Rn], #±<offset_12>**
@ Rd \leftrightarrow M[Rn]
@ Rn \leftarrow Rn \pm <offset_12>

Post-incremento da registro

- **[Rn], ±Rm**
@ Rd \leftrightarrow M[Rn]
@ Rn \leftarrow Rn \pm Rm

Post-incremento da registro scalato

- **[Rn], ±Rm, <sop> #<shift_imm>**
@ Rd \leftrightarrow M[Rn]
@ Rn \leftarrow Rn \pm (Rm <sop> # <shift_imm>)

Modo 2: esempi

11

U
III
52

<code>ldr R2, [R0]</code>	@ R2 \leftarrow M ₃₂ [R0]
<code>ldr R1, [R0,#4]</code>	@ R1 \leftarrow M ₃₂ [R0+4]
<code>ldr R1, [R0], #8</code>	@ R1 \leftarrow M ₃₂ [R0] @ R0 \leftarrow R0+8
<code>ldr PC, [PC, R0, LSL #2]</code>	@ PC \leftarrow M ₃₂ [PC+R0*4]
<code>strb R7, [R9], #1</code>	@ M ₈ [R9] \leftarrow R7 _B @ R9 \leftarrow R9+1
<code>str R5, [R0,#4]!</code>	@ R0 \leftarrow R0+4 @ M ₃₂ [R0] \leftarrow R5

Modo 3 (ind. per HalfWord/signed Byte)

12

U
III
52

sintassi:

STR|LDR[H] Rd, <addressing_mode3>

LDR[SH|SB] Rd, <addressing_mode3>

<addressing_mode3> può essere:

- offset immediato
- offset da registro
- pre-incremento immediato
- pre-incremento da registro
- post-incremento immediato
- post-incremento da registro

differenze rispetto al Modo 2:

- non si possono scalare i registri
- gli offset immediati sono a soli 8bit

Modo 2 e modo 3: tabella riassuntiva

13

U
III
52

	LDR	STR
W	Modo 2	Modo 2
SH	Modo 3	-
H	Modo 3	Modo 3
SB	Modo 3	-
B	Modo 2	Modo 2

NOTA BENE: non ha senso parlare di STORE per quantità con segno, perché non c'è alcuna estensione del segno da effettuare

Modo 4 (ind. per load/store multiplo)

14

U
I
I
52

sintassi:

LDM|STM <addressing_mode4> Rn{!}, <registers>

<addressing_mode4> può essere:

- **IA increment after**
 - start_addr = Rn ; end_addr = Rn + #regs*4 - 4
 - **IB increment before**
 - start_addr = Rn + 4 ; end_addr = Rn + #regs*4
 - **DA decrement after**
 - start_addr = Rn - #regs*4 + 4 ; end_addr = Rn
 - **DB decrement before**
 - start_addr = Rn - #regs*4 ; end_addr = Rn - 4
- (#regs è il numero di registri indicati in <registers>)

! provoca l'aggiornamento del registro Rn: al suo contenuto viene sommata o sottratta la quantità #regs*4

Modo 4 (ind. per load/store multiplo)

15

U
I
I
52

sintassi:

LDM|STM <addressing_mode4> Rn{!}, <registers>

<registers> è la lista di registri da salvare/caricare, racchiusa tra parentesi graffe "{}" e con gli elementi separati da virgola.

esempi:

STMDB SPI, {R0-R7}	@ salva sullo stack i registri @ da R0 a R7
LDMIA SPI, {R0-R7}	@ ricarica R0-R7 salvati @ dall'istruzione precedente
LDMDA R9, {R1,R3-R5}	@ carica i registri R1, R3-R5 @ da M ₃₂ [R9-12].. M ₃₂ [R9]

Classificazione delle istruzioni

16

U
III
52

Elaborazione di dati

- **operazioni aritmetiche e logiche:** ADC, ADD, AND, BIC, EOR, ORR, RSB, RSC, SBC, SUB
- **movimento tra registri:** MOV, MVN
- **confronto:** CMN, CMP, TEQ, TST

Accesso alla memoria

- **load/store tra memoria e registri:** LDR, LDRB, STR, STRB, LDRH, LDRSB, LDRSH, STRH, LDMxx, STMxx

Controllo del flusso

- **branch:** B, Bcc

Istruzioni di elaborazione di dati

17

U
III
52

Queste istruzioni usano la classe di **indirizzamento modo 1**

Regole per le istruzioni di elaborazione di dati:

- tre operandi: due sorgente, uno destinazione
- tutti gli operandi sono a 32 bit, sia registri che costanti
- il risultato è anch'esso a 32 bit ed è posto in un registro

Esempi:

```
add r0, r1, r2      @ r0 = r1 + r2
add r0, r0, r0      @ r0 = r0 + r0 = 2*r0
```

Istruzioni aritmetiche

18

U
U
U
52

Addizione:

ADD{<cond>} Rd, Rn, <addr_model>

Sottrazione:

SUB{<cond>} Rd, Rn, <addr_model>

RSB{<cond>} Rd, Rn, <addr_model>

Moltiplicazione:

MUL{<cond>} Rd, Rn, Rs

(attenzione: MUL ha solo operandi nei registri)

Nessuna istruzione per le divisioni!

- vanno eseguite via software (sottrazioni ripetute);
- SDIV e UDIV presenti solo nei Cortex R e M.

Istruzioni aritmetiche: esempi

19

U
U
U
52

<code>add r0, r1, r2</code>	@ r0 = r1 + r2
<code>adc r0, r1, r2</code>	@ r0 = r1 + r2 + C
<code>sub r0, r1, r2</code>	@ r0 = r1 - r2
<code>sbc r0, r1, r2</code>	@ r0 = r1 - r2 + C - 1
<code>rsb r0, r1, r2</code>	@ r0 = r2 - r1
<code>rsc r0, r1, r2</code>	@ r0 = r2 - r1 + C - 1

- **rsb** sta per *reverse subtraction*
- **C** è il *carry bit* del CPSR
- per impostare i bit di condizione in base al risultato occorre aggiungere il suffisso **s** al simbolo operativo

ADD, SUB, MUL: esempi

20

U
U
U
52

ADD R0, R0, #1	@ R0=R0+1
ADD R0, R0, #-1	@ R0=R0-1
ADDS R0, R0, #-1	@ R0=R0-1 e aggiorna i bit @ di stato (utile nei cicli)
ADD R0, R1, R2, ASR #2	@ R0=R1+R2/4
SUB R0, R0, #1	@ di nuovo R0=R0-1
SUB PC, LR, #4	@ ritorno da interruzione
MUL R0, R1, R2	@ R0=R1*R2

Istruzioni logiche

21

U
U
U
52

AND logico:

AND{<cond>} Rd, Rn, <addr_model>

OR logico:

ORR{<cond>} Rd, Rn, <addr_model>

OR esclusivo:

EOR{<cond>} Rd, Rn, <addr_model>

Bit Clear (Rd=Rn AND NOT <addr_model>):

BIC{<cond>} Rd, Rn, <addr_model>

Istruzioni logiche: esempi

22

U
III
52

<code>and r0, r1, r2</code>	@ r0 = r1 and r2
<code>orr r0, r1, r2</code>	@ r0 = r1 or r2
<code>eor r0, r1, r2</code>	@ r0 = r1 xor r2
<code>bic r0, r1, r2</code>	@ r0 = r1 and not r2

bic sta per *bit clear*: ogni bit 1 nel secondo operando fa azzerare il corrispondente bit nel primo.

AND, ORR, EOR, BIC: esempi

23

U
III
52

<code>AND R0, R1, R2</code>	@ R0=R1 AND R2
<code>AND R0, R1, #0x7</code>	@ estrae da R1 i 3 bit meno @ significativi e li pone in R0
<code>ORR R0, R0, R1</code>	@ R0=R0 OR R1
<code>EOR R0, R1, #0xF</code>	@ R0=R1 XOR 16
<code>EOR R0, R0, R0</code>	@ R0=0
<code>BIC R0, R1, R2</code>	@ R0=R1 AND NOT R2

e il NOT? si può usare **MVN**:

<code>MVN R1, R0</code>	@ R1=NOT R0
<code>MVN R0, R0</code>	@ nega i bit di R0

Move (MOV)

24

U
III
52

Muove un valore tra registri o carica un valore immediato espresso con 12 bit ($\text{imm}_{12} = \text{c}_8 \gg 2 * r_4$)

MOV{<cond>} **Rd**, <addr_model>

MOV R0, #0	@ R0=0
MOV R1, #0xFF00	@ R1=65280
MOV R2, R1	@ R2=R1
MOV R3, R1, LSL #2	@ R3=R1*4
MOV R3, R1, ASR #1	@ R3=R1/2
MOV R1, R1, ROR R2	@ R1 ruotato a destra di R2 bit
MOV PC, LR	@ ritorno da subroutine

Movimento tra registri: esempi

25

U
III
52

mov r0, r2	@ r0 := r2
mvn r0, r2	@ r0 := not r2

mvn sta per *move negated*

Istruzioni di confronto

26

U
III
52

Compare:

aggiorna bit di stato in base al risultato di
 $Rn - \langle \text{addr_mode1} \rangle$:

CMP{<cond>} $Rn, \langle \text{addr_mode1} \rangle$

Test:

aggiorna bit di stato in base al risultato di
 $Rn \text{ AND } \langle \text{addr_mode1} \rangle$:

TST{<cond>} $Rn, \langle \text{addr_mode1} \rangle$

Test Equivalence:

aggiorna bit di stato in base al risultato di
 $Rn \text{ XOR } \langle \text{addr_mode1} \rangle$:

TEQ{<cond>} $Rn, \langle \text{addr_mode1} \rangle$

Operazioni di confronto: esempi

27

U
III
52

<code>cmp r1, r2</code>	@ set cc on $r1 - r2$
<code>cmn r1, r2</code>	@ set cc on $r1 + r2$
<code>tst r1, r2</code>	@ set cc on $r1 \text{ and } r2$
<code>teq r1, r2</code>	@ set cc on $r1 \text{ xor } r2$

I risultati delle operazioni ($-$, $+$, and , xor) non sono salvati in alcun registro;

Solo i bit di condizione (**cc**) del **CPSR** sono modificati da queste istruzioni.

Accesso alla memoria (load/store)

28

UNIVERSITÀ
52

Queste istruzioni usano gli indirizzamenti modo 2 e 3

Load : nel registro destinazione viene caricato il contenuto della locazione di memoria indicata dal modo di indirizzamento

ldr r0, [r1] @ $r0 \leftarrow M_{32}[r1]$

Store : il contenuto del registro destinazione viene salvato nella locazione di memoria indicata dal modo di indirizzamento

str r0, [r1] @ $M_{32}[r1] \leftarrow r0$

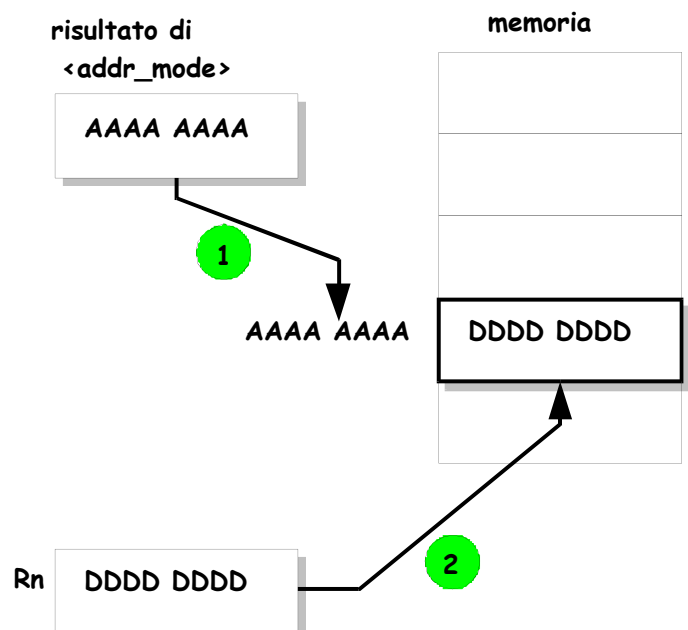
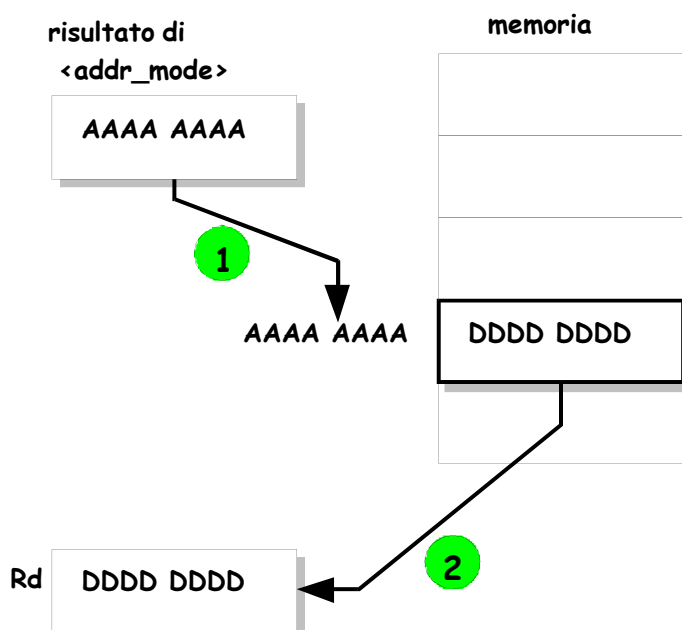
Istruzioni load/store - 2

29

UNIVERSITÀ
52

ldr Rd, <addr_mode2>

str Rn, <addr_mode2>



Istruzione load (LDR)

30

U
III
52

Carica un word da memoria nel registro **Rd**

LDR{<cond>}Rd, <addr_mode2>

Carica in **Rd_{16/8}** un halfword/byte **con zero padding**

LDR{<cond>}H Rd, <addr_mode3>

LDR{<cond>}B Rd, <addr_mode2>

Carica un halfword/byte **con estensione del segno**

LDR{<cond>}SH Rd, <addr_mode3>

LDR{<cond>}SB Rd, <addr_mode3>

LDR: esempi

31

U
III
52

LDR	R0, [R1]	@ r0 ← M ₃₂ [r1]
LDRB	R0, [R1]	@ r0 ← M ₈ [r1]
LDRSH	R0, [R1]	@ r0 ← ext ₃₂ (M ₁₆ [r1])
LDR	R0, [R1, #-8]	@ r0 ← M ₃₂ [r1-8]
LDR	R0, [R1, R2]	@ r0 ← M ₃₂ [r1+r2]

Caricare un valore immediato

32

U
III
52

Non esiste un'istruzione per caricare in un registro un **valore immediato arbitrario!**

Due possibili soluzioni:

- Caricare un valore immediato valido con MOV e poi ottenere quello voluto con somme e shift;

- Usare la **pseudo istruzione**

LDR Rd, =numero

Esempio:

LDR R0, =0x47A0

Pseudoistruzione di estensione di ldr

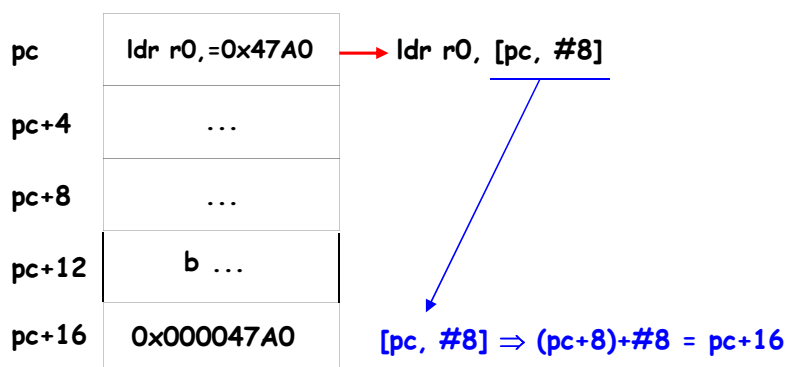
33

U
III
52

La pseudo istruzione **ldr** (estensione di **ldr**) è:

ldr rn, = valore

- l'assemblatore espande la pseudo istruzione, ovvero la realizza tramite una o più istruzioni assembly e aree di supporto;
- viene creato un dato ausiliario, in un luogo "sicuro" nelle vicinanze dell'istruzione, contenente il valore da caricare e vi si accede tramite un indirizzamento relativo al PC



Caricare un indirizzo: ADR

34

U
III
52

Spesso è utile caricare in un registro un indirizzo definito da un **label**

si può usare la **pseudo istruzione**:

ADR Rd, <label>

Esempio:

caricare in R0 l'indirizzo **pippo** al quale inizia un array da elaborare:

ADR R0, pippo

Caricare un indirizzo: ADR, LDR

35

U
III
52

per inserire in un registro un puntatore ci sono due possibilità:

1. si usa la **pseudo istruzione adr**, che permette di caricare in un registro l'indirizzo identificato da un **label**, **purché tale label si trovi nello stesso segmento del codice** (segmento *text*);
2. si usa la **pseudo istruzione** che estende l'istruzione **ldr**, che permette di caricare in un registro l'indirizzo identificato da un **label** ovunque questo sia definito (*text*, *data*, *bss*)

Pseudo istruzione adr

36

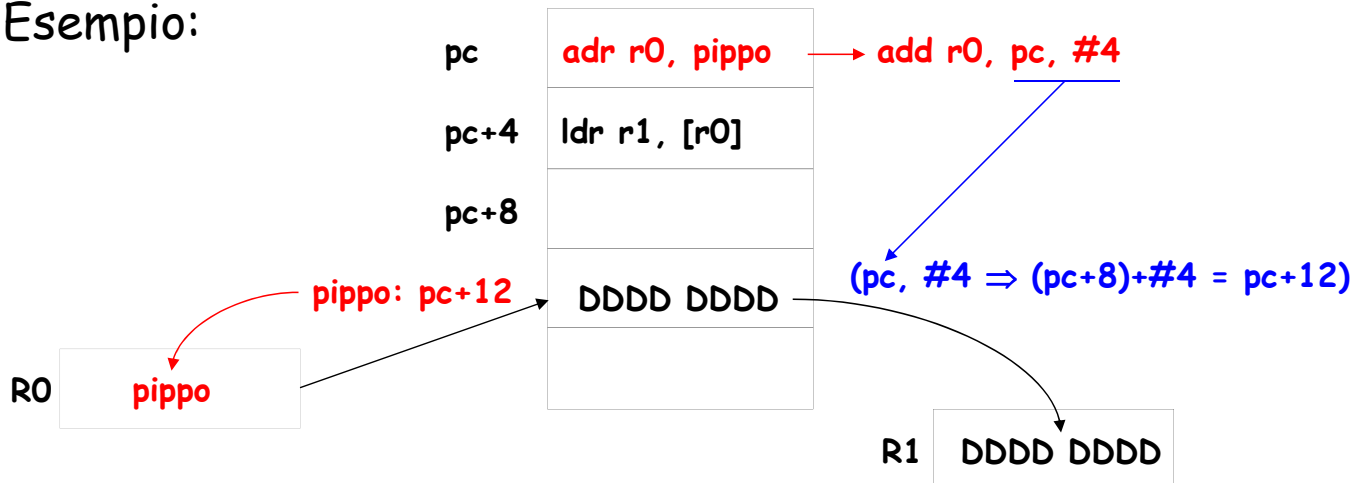
U
I
I
I
52

La sintassi della pseudo istruzione **adr** è:

adr Rd, label

L'assemblatore espande la pseudo istruzione sostituendola con una istruzione **add**, che somma al PC la distanza (offset) dell'indirizzo label: **add Rd, pc, #offset**

Esempio:



Architettura degli Elaboratori

© 2012

Store (STR)

37

U
I
I
I
52

Salva **Rn** in un word di memoria

STR{<cond>}Rn, <addr_mode2>

Salva **Rn_{16/8}** in un halfword/byte di memoria

STR{<cond>}H Rn, <addr_mode3>

STR{<cond>}B Rn, <addr_mode2>

Salva un halfword/byte con estensione del segno:

non c'è; non ha senso estendere il segno in memoria

Architettura degli Elaboratori

© 2012

STR: esempi

38

U
=

STR R0, [R1, #4]!	@ preincremento: $R1=R1+4$, @ poi $M_{32}[R1] \leftarrow R0$
STR R0, [R1], #4	@ postincremento: @ $M_{32}[R1] \leftarrow R0$, poi $R1=R1+4$
STRB R0, [R1, -R2]	@ offset da registro: @ $M_8[R1-R2] \leftarrow R0_B$
STRH R0, [R1, #8]	@ offset immediato: @ $M_{18}[R1+8] \leftarrow R0_W$

Load/store multipli: LDM/STM

39

U
=

Queste istruzioni usano l'indirizzamento modo 4

LDM<xx> Rn{!}, {lista registri}

carica un insieme di registri da memoria a partire da $M_{32}[Rn]$

STM<xx> Rn{!}, {lista registri}

salva un insieme di registri in memoria (da $M_{32}[Rn]$)

<xx> indica come aggiornare Rn:

- **IB**: Increment Before
- **IA**: Increment After
- **DB**: Decrement Before
- **DA**: Decrement After

LDM/STM e lo stack

40

D
I
I
I
52

4 tipi di stack: **FA**, **FD**, **EA**, **ED**

F = **full**: SP punta all'ultimo elemento **usato**

E = **empty**: SP punta al primo elemento **libero**

A = **ascending**: SP **cresce** con push sullo stack

D = **descending**: SP **decresce** con push sullo stack

valgono le seguenti equivalenze:

LDM**IB** = LDM**ED**

STM**IB** = STM**FA**

LDM**IA** = LDM**FD**

STM**IA** = STM**EA**

LDM**DB** = LDM**EA**

STM**DB** = STM**FD**

LDM**DA** = LDM**FA**

STM**DA** = STM**ED**

LDM/STM: esempi di accesso allo stack

41

D
I
I
I
52

Di solito si usa **Rn = R13 = SP**

Nella convenzione usata nel PD32 lo stack pointer

- punta all'ultimo elemento inserito (**F**)
- cresce per indirizzi decrescenti (**D**):

STM**FD** SP!, {R0-R7}

@ salva sullo stack i registri
@ da R0 a R7 (come STMDB)

LDM**FD** SP!, {R0-R7}

@ ricarica R0-R7 salvati
@ dall'istruzione precedente
@ (come LDMIA)

LDMDB R12, {R12, SP, PC}

@ carica i registri R12, SP, PC
@ con $M_{32}[R12-12]$, $M_{32}[R12-8]$,
@ e $M_{32}[R12-4]$

Esempio: primo programma

42

Si costruisce ora un primo programma in linguaggio assembly per processori ARM:

Si voglia spostare il contenuto di una serie di locazioni contigue di memoria da una posizione ad un'altra (programma che copia il contenuto di un array

U
III
52

```
copy:  adr R1, TABLE1 @ R1 punta a TABLE1
        adr R2, TABLE2 @ R2 punta a TABLE2
        ldr R0, [R1] @ carica il primo valore
        str R0, [R2] @ copia il primo valore
        ...
        ...
TABLE1: ... @ array sorgente
        ...
TABLE2: ... @ array destinazione
```

Esempio: somma

43

Bisogna ora continuare a copiare gli elementi successivi:

```
copy:  adr R1, TABLE1 @ R1 punta a TABLE1
        adr R2, TABLE2 @ R2 punta a TABLE2
        ldr R0, [R1] @ carica il primo valore
        str R0, [R2] @ copia il primo valore
        add R1, R1, #4 @ R1 punta al successivo
        add R2, R2, #4 @ R2 punta al successivo
        ldr R0, [R1] @ carica il secondo valore
        str R0, [R2] @ ... e copialo
```

U
III
52

Si sono incrementati i due puntatori alle tabelle (i registri R1 e R2) della dimensione degli elementi (word da 4 byte)

Esempio: offset

44

D
III
52

È possibile conglobare l'operazione di incremento dei puntatori nelle istruzioni di load e store, usando l'indirizzamento a due componenti (registro+offset immediato):

copy:	adr R1, TABLE1	@ R1 punta a TABLE1
	adr R2, TABLE2	@ R2 punta a TABLE2
	ldr R0, [R1]	@ carica il primo val.
	str R0, [R2]	@ salva il primo valore
	ldr R0, [R1, #4]	@ carica il secondo val.
	str R0, [R2, #4]	@ ... e copialo

Usati i puntatori contenuti in R1 e R2, si è aggiunta ad essi la dimensione dell'elemento (4) in modo che puntino al successivo (con un'unica istruzione):

ldr R0, [R1, #4] @ R0 ← M₃₂[R1+4]

Esempio: pre-incremento

45

D
III
52

Nell'ultima versione del programma i registri R1 e R2 continuano a puntare all'inizio delle rispettive tabelle; per scorrerle tutte bisogna aumentare, ogni volta, gli offset (8, 12, 16, ...), oppure (meglio) si incrementano i puntatori R1 e R2, usando l'indirizzamento con pre-incremento:

copy:	adr R1, TABLE1	@ R1 punta a TABLE1
	adr R2, TABLE2	@ R2 punta a TABLE2
	ldr R0, [R1]	@ carica il primo valore
	str R0, [R2]	@ salva il primo valore
loop:	ldr R0, [R1, #4]!	@ carica il secondo val.
	str R0, [R2, #4]!	@ ... e copialo

Il punto esclamativo comporta la pre-modifica del registro:

ldr R0, [R1, #4]! @ R1 ← R1+4, R0 ← M₃₂[R1]

Esempio: post-incremento

46

Per scorrere tutta la tabella basta ripetere invariate le ultime due istruzioni; copiato il primo elemento, si può procedere (loop) con i successivi.

È possibile uniformare anche la prima operazione, usando l'indirizzamento con post-incremento:

copy:	adr R1, TABLE1	@ R1 punta a TABLE1
	adr R2, TABLE2	@ R2 punta a TABLE2
loop:	ldr R0, [R1], #4	@ carica dal 1.mo vettore
	str R0, [R2], #4	@ salva nel 2.ndo vettore

Rispetto al caso precedente, R1 e R2 sono prima usati per accedere agli operandi, e poi vengono incrementati di 4:

ldr R0, [R1], #4 @ R0←M₃₂[R1], R1←R1+4

Servono altre istruzioni per eseguire il loop il numero di volte desiderato (istruzioni di confronto e di salto)

Istruzioni di controllo del flusso (salto)

47

Salto:

B{<cond>} <indirizzo>

Salto a subroutine (salva PC in LR):

BL{<cond>} <indirizzo>

L'indirizzamento è di tipo auto-relativo: ±32MB dalla posizione attuale

Chiamata a sistema (interruzione sw):

SWI{<cond>} <immed_24>

Istruzioni di salto: esempi

48

U
I
I
I
52

B (branch incondizionato)

- B <indirizzo> @ PC←indirizzo

BL (branch and link incondizionato)

- BL <indirizzo> @ LR←PC-4, PC←indirizzo

```
b label1 @ salta a label1
label1:  b label2 @ salta e salva il PC
cont:   ... @ continua il programma
label2:  ... @ fa qualcosa (subroutine)
        mov PC, LR @ torna a cont
```

Condizioni

49

U
I
I
I
52

Estensione mnemonica	Significato	Flag di condizione	Opcode [31:28]
EQ	Uguali	Z=1	0000
NE	Non uguali	Z=0	0001
CS/HS	Carry Attivato / Senza segno maggiore o uguale	C=1	0010
CC/LO	Carry Disattivato / Senza segno minore	C=0	0011
MI	Negativo	N=1	0100
PL	Positivo o Zero	N=0	0101
VS	Overflow	V=1	0110
VC	Non Overflow	V=0	0111
HI	Senza segno maggiore	C=1 e Z=0	1000
LS	Senza segno minore o uguale	C=0 o Z=1	1001
GE	Con segno maggiore o uguale	N=V	1010
LT	Con segno minore	N!=V	1011
GT	Con segno maggiore	Z=0 e N=V	1100
LE	Con segno minore o uguale	Z=1 o N!=V	1101
AL	Sempre (è il default)	-	1110
NV	Mai	-	1111

Istruzioni condizionate

50

U
=
=
52

Tutte le istruzioni di elaborazione di dati e di salto possono essere rese condizionate, ovvero la loro effettiva esecuzione può essere fatta dipendere dallo stato dei bit ZNCV nel CPSR.

Per rendere una istruzione condizionata è sufficiente apporre come suffisso dell'istruzione il codice della condizione.

Esempi:

bne label	@ salta se non uguale
addlt r0, r1, #10	@ somma se minore o uguale
movcc R1, #6	@ copia se non c'è stato riporto

Confronto e salto: esempi

51

U
=
=
52

CMP R1, #0	@ confronta R1 e 0
BEQ label1	@ salta a label1 se R1=0
CMP R1, R0	@ confronta R1 e R0
BGT label2	@ salta a label2 se R1>R0 (signed)
TST R1, #0x42	@ controlla i bit 1 e 6 di R1
BLNE errore	@ salta alla subroutine errore se @ almeno uno di tali bit è ≠0
SWI #0x0F	@ chiama la funzione di sistema @ numero 16

Completamento dell'esempio

52

U
III
52

Supponiamo che l'array da copiare contenga 16 elementi: le istruzioni scritte prima vanno eseguite ripetutamente (loop) 16 volte; allo scopo si può utilizzare un registro (ad es. R3) come contatore:

- vi si inserisce il valore iniziale 16,
- lo si decrementa di una unità ad ogni iterazione,
- con una istruzione di salto condizionato si riesegue il loop se il contatore è diverso zero, altrimenti no:

```
copy:  mov R3, #0x10 @ valore iniziale contatore
      adr R1, TABLE1 @ R1 punta a TABLE1
      adr R2, TABLE2 @ R2 punta a TABLE2
loop:  ldr R0, [R1], #4 @ carica dal primo vettore
      str R0, [R2], #4 @ salva nel secondo vettore
      subs R3, R3, #1 @ decrementa il contatore
      bne loop @ salta se non è zero
```

Fine

03.b

U
III

ARM: le istruzioni assembly

