Sintassi delle istruzioni ARM

Argomento:

- Istruzioni condizionate
- Classi di sintassi
- Miscellanea: pseudo-istruction

Quick reference card

- ARM Quick Reference Card: tabella che riassume tutte le istruzioni ARM e la loro rappresentazione assembly

- In moodle:
  - Quick reference card
  - Video che spiega come usare il documento

Istruzioni condizionate

Registro di stato CPSR

- Di default, solo le istruzioni di confronto modificano sempre i bit NZCV.
  - Altre operazioni non modificano questi bit.
  - Istruzioni di confronto: CMP, TST, TEQ, CMN,…
- Una generica istruzione modifica i bit di stato solo quando si aggiunge il suffisso S al simbolo mnemonico dell'istruzione:
  - Nella codifica binaria dell'istruzione, il 20° bit viene messo ad 1

```
ADD  R2, R0, R1 @ non modifica bit NZCV
ADDS R2, R0, R1 @ modifica bit NZCV
```

Istruzioni condizionate

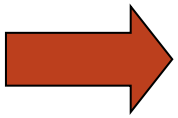
- Tutte le istruzioni ARM, comprese quelle di salto, possono essere rese condizionate:
  - La loro esecuzione può essere fatta dipendere dallo stato dei bit ZNCV nel CPSR impostati da una precedente istruzione.
- Per rendere una istruzione condizionata è sufficiente apporre come suffisso dell'istruzione assembly il codice della condizione.
- Nella codifica binaria dell'istruzione, il 20° bit viene messo ad 1

Esempi di uso

- Permettono di velocizzare il codice
  - risparmiando salti condizionati
  - mantenendo pieni i pipeline
  - Nessun branch misprediction

```
if (a > 10)
      a = 0;

else
      a = 1;
```



```
CMP    R0, #10
MOVGT  R0, #0
MOVLE  R0, #1
```

Lista delle condizioni

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

Esempi con confronto e salto

```
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 =1
```

Esempi con altre istruzioni

- `MOVEQ R0, #0`
  esegui se uguali → Z=1
- `MOVNE R0, #0`
  esegui se non uguali → Z=0
- `MOVGT R0, #0`
  esegui se maggiore (con segno) → Z=0 e N=V
- `MOVHI R0, #0`
  esegui se maggiore (senza segno) → Z=0 e C=1
- `MOVAL R0, #0`
  esegui sempre

Sintassi delle istruzioni

Sintassi delle istruzioni

- Vediamo 3 classi di istruzioni che hanno una sintassi simile

- ARM è un'architettura RISC:
  - niente indirizzamento assoluto: non usa puntatori in memoria, ma solo nei registri
  - indirizzamento in memoria (indiretto con registro) solo per le istruzioni LDR (load) e STR (store)

Classi di sintassi

Vediamo 3 classi di istruzioni che hanno una sintassi simile:

- Classe 1: per istruzioni di elaborazione dati

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

- Classe 2: per Load&Store di word o unsigned byte

  `LDR, LDRB, STR, STRB`

- Classe 3: per L&S di halfword o signed byte

  `LDRH, LDRSB, LDRSH, STRH`

Classe 1: operazioni a 3 indirizzi

Istruzioni di elaborazione dati a 3 indirizzi: ADC, ADD, AND, BIC, EOR, ORR, RSB, RSC, SBC, SUB

<opcode>{<cond>}{S} <Rd>, <Rn>, <shifter_operand>

- <cond>: stabilisce l'esecuzione condizionata
- S: stabilisce se modifica i bit di condizione
- <Rd>: destinazione (Rd è sempre specificato per primo)
- <Rn>: primo operando
- <shifter_operand>: secondo operando

Esempio: ADD R4, R3, R2 @R4 ← R3+R2

Classe 1: operazioni a 2 indirizzi

Istruzioni di elaborazione dati a 2 indirizzi: `CMN, CMP, MOV, MVN, TEQ, TST`

`<opcode>{<cond>}{S} <Rd>, <shifter_operand>`

- `<cond>`: stabilisce l'esecuzione condizionata
- `S`: stabilisce se modifica i bit di condizione
- `<Rd>`: destinazione (Rd è sempre specificato per primo)
- `<Rn>`: primo operando
- `<shifter_operand>`: secondo operando

Esempio: `MOV R0, #0` `@R0` $\leftarrow$ 0

Campo: <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

Indirizzamento immediato con <shifter_operand>

Il campo dell'istruzione contiene il valore su cui operare

Esempio: `ADD R3, R3, #1  @ R3 ← R3+1`

12 bit a disposizione per l'operando immediato, con la seguente struttura:

- 8 bit (bit c) definiscono un valore c ($0 \leq c \leq 255$);
- 4 bit (bit r) specificano una rotazione verso destra di 2r posizioni

The image part with relationship ID rId3 was not found in the file.

Indirizzamento di registro con <shifter_operand>

Il campo dell'istruzione contiene il registro con il valore

Esempio: ADD R2, R3, R4 @ R2 ← R3+R4

Non è possibile l'indirizzamento indiretto di registro con la classe 1

- Le istruzioni di elaborazione dati non accedono alla memoria (tipico dei RISC)

indirizzamento di registro con `<shifter_operand>` con shift

- I tipi di shift sono:
  - LSL: logico a sinistra
  - LSR: logico a destra
  - ASR: aritmetico a destra
  - ROR: rotazione a destra
  - RRX = 33 bit rotazione a destra right usando il bit C come 33° bit
- Il valore di shift può essere shiftato di una quantità:
  - specificata in modo immediato (0..31: 5 bit): `Rm, LSR #shift_imm`
  - oppure specificata da un altro registro (Rs): `Rm, LSR Rs`

indirizzamento di registro con <shifter_operand> con shift (2)

```
ADD R3, R1, R2, LSL #2
          @ R3 ← R1+R2*4

ADD R3, R1, R2, ASR R5
          @ R3 ← R1+R2/2^R5
```

Classe 1: esempi

```
MOV R0, #0          @ R0 ← 0

ADD R3, R3, #1      @ R3 ← R3+1

CMP R7, #1000       @ cc ← (R7-1000)

BIC R9, R8, #0xff00  @ R9 ← R8 and not 0xff00

MOV R2, R0          @ R2 ← R0

ADD R4, R3, R2      @ R4 ← R3+R2
```

Classe 1: esempi (2)

```
MOV R2, R0, LSL #2        @ R2 ← R0*4

ADD R9, R5, R5, LSL #3    @ R9 ← R5+R5*8 = R5*9

SUB R9, R5, R5, ASR #3    @ R9 ← R5-R5/8

RSB R9, R5, R5, ASR #3    @ R9 ← R5/8-R5

MOV R5, R3, RRX       @ R5 ← R3 ruotato esteso a
                      @   destra di una posizione

MOV R7, R4, ROR R3    @ R7 ← R4 ruotato a destra di
                      @   R3 posizioni
```

Classi 2 e 3: load and store

LDR: nel registro destinazione viene caricato la word della locazione di memoria indicata dal modo di indirizzamento

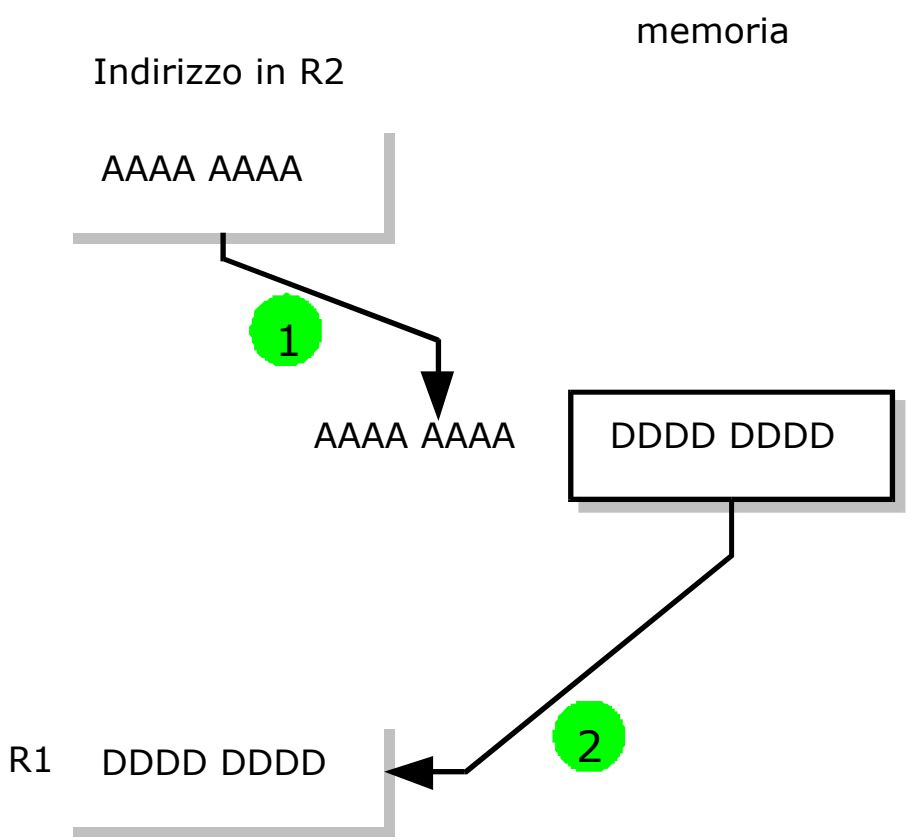
LDR R0, [R1] @ R0 ← M32[R1]

STR: la word del registro sorgente viene salvato nella locazione di memoria indicata dal modo di indirizzamento

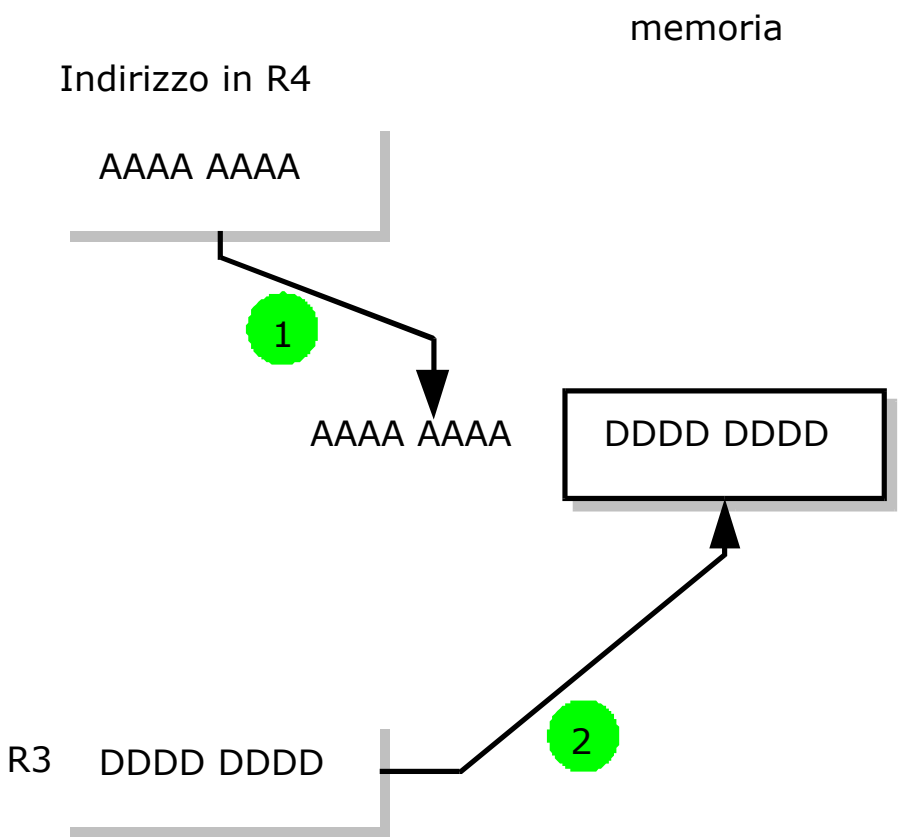
STR R0, [R1] @ M32[R1] ← R0

Load and store

```
LDR R1, [R2]
```




```
STR R3, [R4]
```



LDR/STR di byte/halfword

- LDR/STR legge/scrive una word (32 bits) in un registro di una word
- Si possono leggere/scrivere byte/halfword con/senza segno aggiungendo i suffissi B, H, SB, SH

`LDRB, LDRH, LDRSB, LDRSH`

`STRB, STRH`

- LDRSB, LDRSH: il byte/halfword vengono copiati in un registro 32 con estensione di segno
- LDRB, LDRH: il byte/halfword vengono copiati in un registro 32: i 24/16 bit più significativi vengono impostati a 0
- Con STR, non si usano SB/SH: non ha senso estendere il segno in memoria

LDR di byte/halfword

Istruzione	Legge	Estende a 32 bit	Esempio
LDRB R1, [R2]	1 byte (8 bit)	Inserisce 0	M8[R2] = 0xAA → R1 = 0x000000AA
LDRH R1, [R2]	1 halfword (16 bit)	Inserisce 0	
LDRSB R1, [R2]	1 byte (8 bit)	Inserisce bit di segno	M[R2] = 0xAA → R1 = 0xFFFFFFAA M[R2] = 0x4A → R1 = 0x0000004A
LDRSH R1, [R2]	1 halfword (16 bit)	Inserisce bit di segno	

STR di byte/halfword

Istruzione	Legge	Esempio
STRB R1, [R2]	1 byte (8 bit)	R2 = 0xAABBCCDD → M8[R1] = 0xDD
STRH R1, [R2]	1 halfword (16 bit)	R2 = 0xAABBCCDD → M16[R1] = 0xCCDD

Classe 2 per Load/Store di word/unsigned Byte

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

`<addressing_mode2>` è un indirizzamento di registro indiretto con un eventuale:

- senza offset (indirizzamento indiretto di registro)
- 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

<addressing_mode2> senza offset

Il valore dell'operando è puntato da un registro Rn:

- Rn contiene l'indirizzo dell'operando

- Implementa l'indirizzamento di registro indiretto
- Sintassi: [Rn]

Esempio:

```
LDR R1, [R0]    @ R1 ← M[R0]
```

<addressing_mode2> con offset

- Offset immediato

[Rn, #±<offset_12>] @ Rd ↔ M[Rn ± <offset_12>]

- Offset da registro

[Rn, ±Rm] @ Rd ↔ M[Rn ± Rm]

- Offset da registro scalato

[Rn, ±Rm, <sop> #<shift_imm>]

@ Rd ↔ M[Rn ± (Rm <sop> # <shift_imm>)]

Esempi con offset

- Offset immediato:

```
LDR R1, [R0,#4]      @ R1 ← M32[R0+4]
```

- Offset da registro:

```
STR R2, [R0, R1]     @ M32[R0+R1] ← R2
```

- Offset da registro scalato:

```
LDR PC, [PC, R0, LSL #2]

                     @ PC ← M32[PC+R0*4]
```

<addressing_mode2> con pre-incremento

- Pre-incremento immediato

  [Rn, #±<offset_12>]!

  @ Rn ← Rn ± <offset_12>

  @ Rd ↔ (Rn)

- Pre-incremento da registro

  [Rn, ±Rm]!

  @ Rn ← Rn ± Rm

  @ Rd ↔ (Rn)

- Pre-incremento da registro scalato

  [Rn, ±Rm, <sop> #<shift_imm>]!

  @ Rn ← Rn ± (Rm <sop> #<shift_imm>)

  @ Rd ↔ (Rn)

Rispetto al caso con offset, compare il ! che forza la scrittura in Rn

Esempi con pre-incremento

- Pre-incremento immediato:

```
STR R5, [R0,#4]!      @ R0 ← R0+4
                      @ M32[R0] ← R5
```

- Pre-incremento da registro:

```
LDR R2, [R0, R1]!     @ R0 ← R0 + R1
                      @ R2 ← M32[R0]
```

- Offset da registro scalato:

```
LDR R4, [R0, R1, LSL #2]! @ R0 ←R0+R1*4
                          @ R4 ← M32[R0]
```

<addressing_mode2> con post-incremento

- Post-incremento immediato

  `[Rn], #±<offset_12>`

  `@ Rd ↔ M[Rn]`

  `@ Rn ← Rn ± <offset_12>`

- Post-incremento da registro

  `[Rn], ±Rm`

  `@ Rd ↔ M[Rn]`

  `@ Rn ← Rn ± Rm`

- Post-incremento da registro scalato

  `[Rn], ±Rm, <sop> #<shift_imm>`

  `@ Rd ↔ M[Rn]`

  `@ Rn ← Rn ± (Rm <sop> # <shift_imm>)`

Esempi con post-incremento

- Post-incremento immediato:

```
STR R1, [R0], #-8     @ M32[R0] ← R1
                      @ R0 ← R0-8
```

- Post-incremento da registro:

```
LDR R2, [R0], R1      @ R2 ← M32[R0]
                      @ R0 ← R0 + R1
```

- Offset da registro scalato:

```
LDR R4, [R0], R1, ASR #2   @ R4 ← M32[R0]
                           @ R0 ←R0+R1/4
```

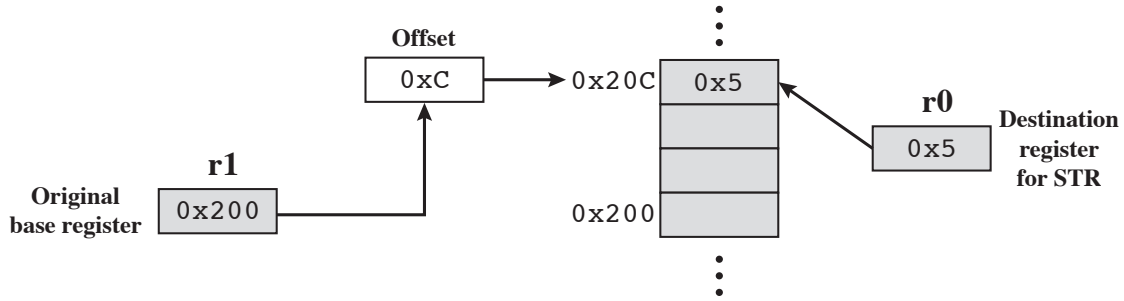
Pre/post-incremento con registro scalato

L'offset nel registro Rm può essere fatto scorrere di un numero di passi specificato con valore immediato da 5 bit

Pre-incremento	Post-incremento
[Rn, ±Rm, LSL #num]!	[Rn], ±Rm, LSL #num
[Rn, ±Rm, LSR #num]!	[Rn], ±Rm, LSR #num
[Rn, ±Rm, ASR #num]!	[Rn], ±Rm, ASR #num
[Rn, ±Rm, ROR #num]!	[Rn], ±Rm, ROR #num
[Rn, ±Rm, RRX]!	[Rn], ±Rm, RRX

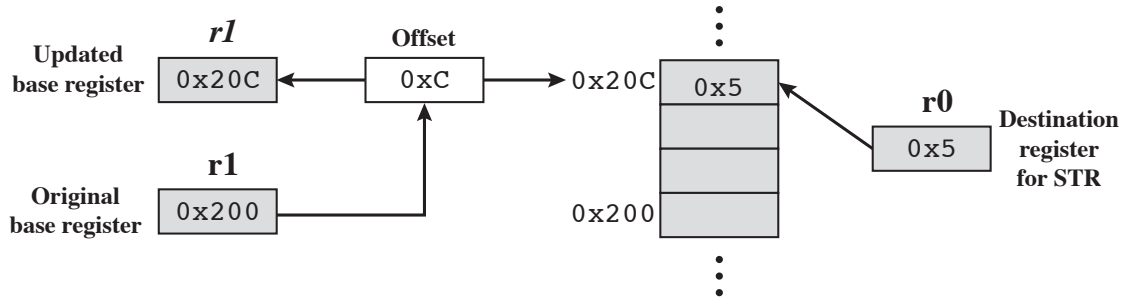
Esempio: LDR R1, [R0], -R2, ASR #2
@ R1←M_{32}[R0]; R0←R0-(R2/4)

```
STRB r0, [r1, #12]
```



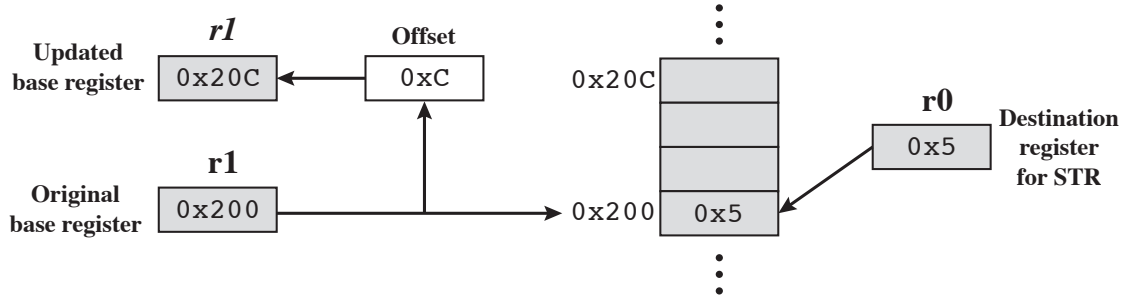
(a) Offset

```
STRB r0, [r1, #12]!
```



(b) Preindex

```
STRB r0, [r1], #12
```



(c) Postindex

Classe 3 per HalfWord/signed Byte

```
STR|LDR[H] Rd, <addressing_mode3>
LDR[SH|SB] Rd, <addressing_mode3>
```

<addressing_mode3> è un indirizzamento di registro con un eventuale:

- offset immediato (solo 8 bit)
- offset da registro
- pre-incremento immediato
- pre-incremento da registro
- post-incremento immediato
- post-incremento da registro

Differenze rispetto alla Classe 2:

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

Classe 2 e classe 3: tabella riassuntiva

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

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

Tecniche di indirizzamento in ARM

Immediato: ADD R1, R2, #0xAA

~~Diretto di memoria~~

~~Indiretto di memoria~~

Diretto di registro: ADD R1, R2, R3

Indiretto di registro: LDR R1, [R3]

Con offset: LDR R1, [R3, #8]

Autorelativo: LDR R1, [PC, #-8]

Miscellanea

Caricare un valore immediato: LDR

- Non esiste un'istruzione per caricare in un registro un valore immediato o indirizzo arbitrario!

- Due possibili soluzioni:
  1. Caricare un valore immediato valido con MOV e poi ottenere quello voluto con somme e shift;
  2. Usare la pseudo istruzione

```
LDR Rd, =numero
```

Esempio:

```
LDR R0, =0x47A0
```

Pseudo istruzione di estensione di ldr

- `LDR rn, = valore` è una pseudo istruzione che estende ldr
  - l'assemblatore espande la pseudo istruzione tramite una o più istruzioni macchina
  - 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

Pseudo istruzione di estensione di ldr (2)