Strutture di dati in assembly ARM

Argomento:

- Array e matrici
- Stack
- Liste concatenate

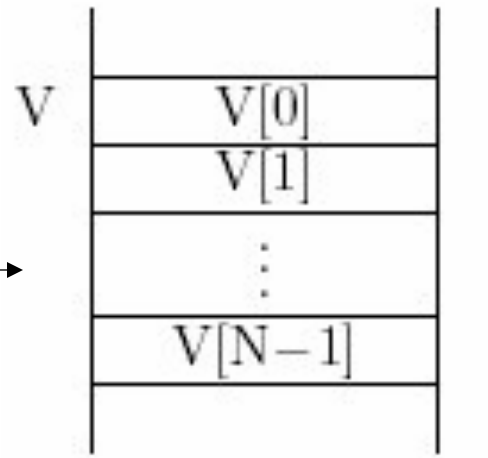
Accesso alle strutture di dati

- A livello di linguaggio assembly, c'è la completa visibilità di come sono organizzate in memoria le strutture di dati con i loro elementi.
- Esaminiamo le più comuni strutture dati e come si accede ai loro elementi:
  - Array
  - Matrici
  - Stack
  - Liste concatenate

Array

- La definizione di un array si effettua specificandone le seguenti informazioni:
  - L'**indirizzo iniziale** in memoria V,
  - il **numero degli elementi** N,
  - la **lunghezza L di ciascun elemento** (in byte).

Organizzazione in memoria di un vettore di N elementi

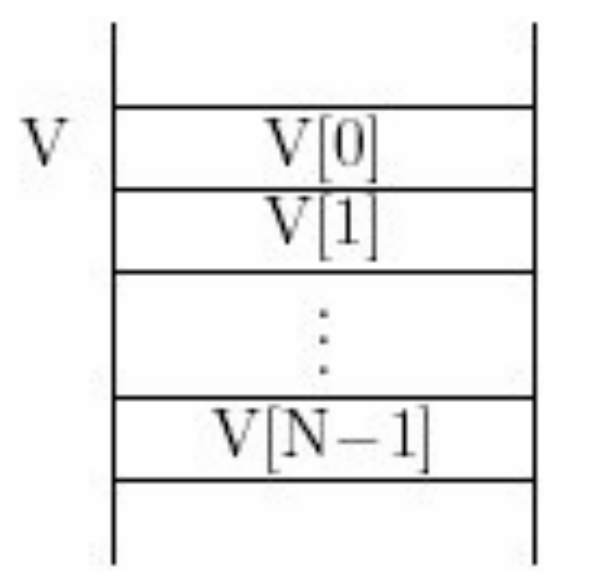


Definizione di un array

Un array di alloca con le istruzioni:

```
        .BSS
V:  .SKIP N*L
```

- .BSS (dati non inizializzati)
- una direttiva label (V:), che definisce l'indirizzo iniziale dell'area di memoria;
- una direttiva (.SKIP o .SPACE), che ne definisce l'estensione (in byte)



Accesso agli elementi di un array

Per accedere agli elementi di un array, si può usare un metodo di **indirizzamento a due componenti** (ad es. con registro indice):

- Una componente fissa costituita dall'indirizzo iniziale V
- Una componente variabile che fornisce l'offset dell'elemento rispetto a V.

Accesso agli elementi di un array (2)

- Se l'indice i parte da zero, l'indirizzo dell'elemento V[i] (ind V[i]) è

Ind V[i] = V + offset

offset = i*L:

- Se l'indice i parte da uno, l'indirizzo dell'elemento V[i] (ind V[i]) è

Ind V[i] = V + offset

offset = (i-1)*L:

- È più conveniente che l'indice i parta da zero.

Accesso agli elementi di un array (3)

Se L è potenza intera di 2, allora il calcolo dell'offset (i*L) può essere ottenuto con una semplice operazione di scorrimento verso sinistra

Esempio: Si voglia copiare in R3 il valore V[i]; L=4 (V vettore di word)

```
LDR R0, =V              @indirizzo di V in R0
MOV R1, #i              @indice i in R1
LSL R1, R1, #2          @ Calcola offset i*4
LDR R3, [R0, R1]        @ Copia V[i] in R3
```

Si possono unire con:
`LDR R3, [R0, R1, LSL #2]`

Accesso agli elementi di un array (4)

Quando L non è una potenza intera di 2, si calcola il prodotto i*L con MUL

- Una moltiplicazione è più costosa (in tempo di esecuzione) di una somma o di uno shift.

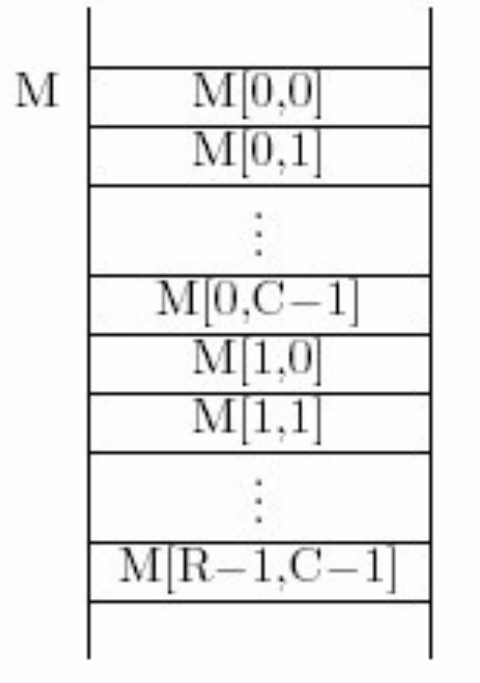
Esempio: Si voglia copiare in R3 il valore V[i];

```
LDR R0, =V          @ indirizzo di V in R0
MOV R1, #i          @ indice i in R1
MOV R2, #L          @ parametro L in R2
MUL R1, R1, R2      @ Calcola offset i*L
LDR R3, [R0, R1]    @ Copia V[i] in R3
```

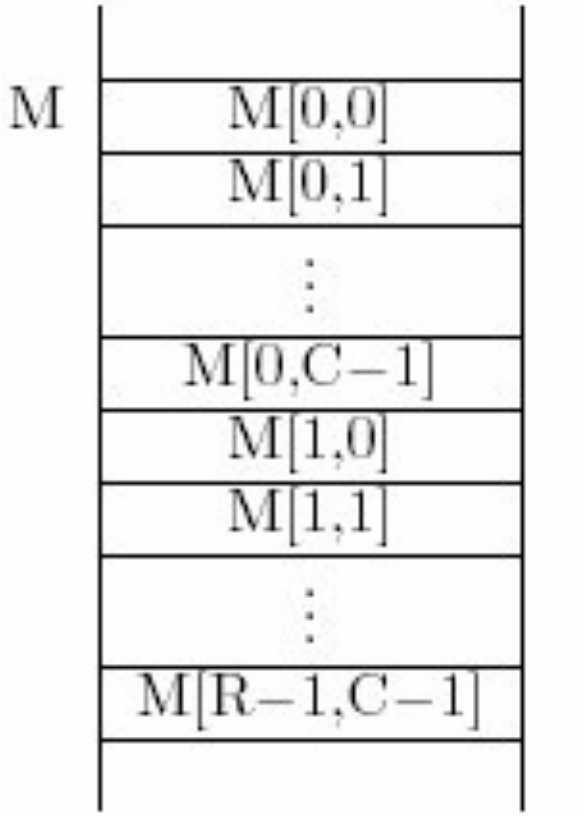
Matrici

Gli elementi di una matrice possono essere collocati in memoria ordinati per riga oppure per colonna.

Organizzazione per righe di una matrice con R righe e C colonne



Definizione di una matrice



- La definizione di una matrice richiede le seguenti informazioni:
  - L'**indirizzo iniziale** M
  - Il **numero di righe** R
  - Il **numero di colonne** C
  - La **lunghezza L di ciascun elemento**.
- L'allocazione si effettua con le istruzioni:

```
        .BSS
M:      .SKIP R*C*L
```

Accesso agli elementi di una matrice

- Per individuare un elemento si usano due indici: riga i e colonna j
- L'indirizzo dell'elemento M[i,j] è dato da:

ind M[i,j] = M + (i*C+j)*L

- Conviene usare un metodo di indirizzamento a due componenti
  - Indirizzo base costituito da M
  - Offset calcolato valutando l'espressione (i*C+j)*L

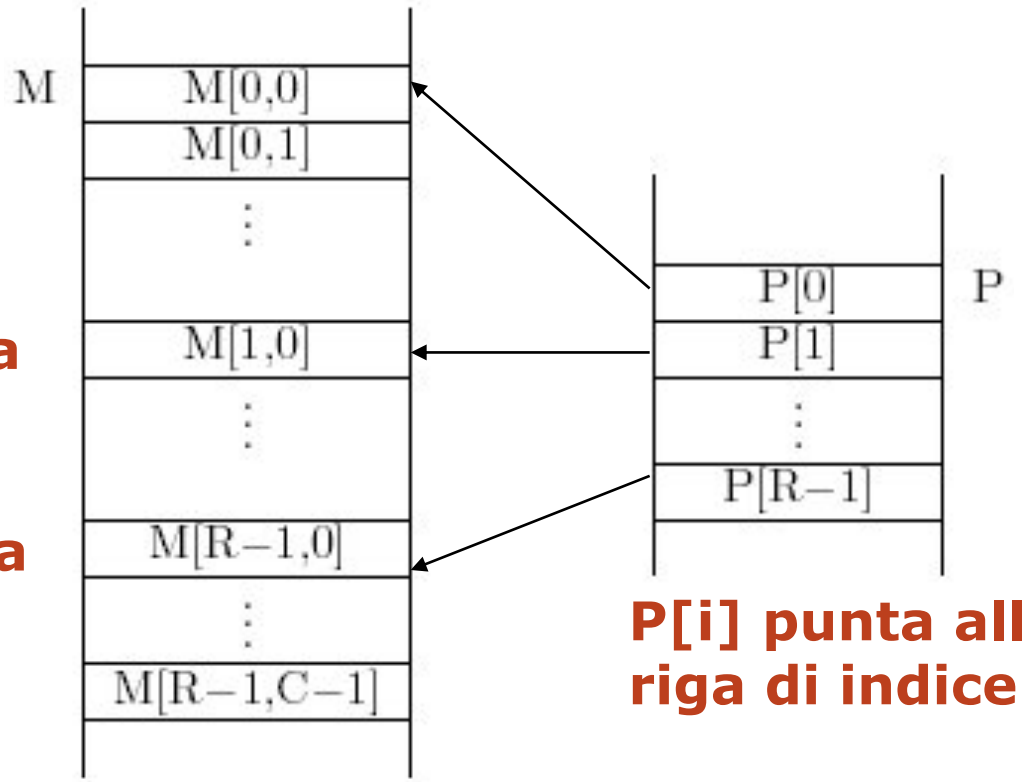
Accesso agli elementi di una matrice (2)

- Si voglia copiare R0 in M[i,j]:
  - i sia contenuto in R1, j in R2, e M in R4.
  - Vettore di halfword (L=2)

```
LDR R3, =C              @ Copia C in R3
MUL R3, R1, R3          @ Calcola R3 = i*C
ADD R3, R3, R2          @ Calcola R3 = i*C+j
LSL R3, R3, #1          @ Calcola R3 = (i*C+j)*2
STRH R0, [R4, R3]       @ Copia R0 in M[i,j]
```

Matrice con array ausiliario di puntatori

Utilizzando un **array ausiliario di puntatori** si evita il calcolo del prodotto i*C e l'accesso alla matrice si riconduce a 2 accessi ad array.



inizio 2ª riga

inizio ultª riga

P[i] punta all'inizio della riga di indice i.

Accesso con array ausiliario di puntatori

Definizione della matrice M e dell'array ausiliario P:

```
.BSS
M:   .skip R*C*L

.DATA
P: .word M, M+C*L, M+2*C*L, …, M+(R-1)*C*L
```

Accesso con array ausiliario di puntatori (2)

Con L=2, i in R1, j in R2, P in R4, il trasferimento R0 $\rightarrow$ M[i,j] si può ottenere così:

```
LSL R3, R1, #2              @ i*4 (P contiene
                            @ indirizzi)
LDR R3, [R4, R3]            @ Indirizzo riga i
                            @ in R3
ADD R3, R3, R2, LSL #1 @ Indirizzo di M[i,j]

STRH R0, [R3]               @ Copia R0 in M[i,j]
```

Accesso con array ausiliario di offset

- Al posto dell'array di puntatori P, si può usare **un array ausiliario O con gli offset di ciascuna riga** rispetto all'indirizzo iniziale M.

```
.BSS
M:    .skip    R*C*L
.DATA
O:    .hword   0, C*L, 2*C*L, …, (R-1)*C*L
```

- Mentre i puntatori dell'array P sono da 4 byte, gli offset di O possono spesso essere da 2 o da 1 byte;
- Un unico array O di offset può essere utilizzato per tutte le matrici dello stesso tipo (stesso numero di righe e di colonne e stessa lunghezza degli elementi).

Accesso con array ausiliario di offset (2)

Usando l'array O di offset, con L=2, i in R1, j in R2, M in R3, O in R4, il trasferimento R0 → M[i,j] si può ottenere così:

```
LSL  R5, R1, #1      @ i*2 (O contiene hwords)
LDRH R4, [R4, R5]    @ Offset riga i in R4
ADD  R4, R4, R2, LSL #1  @ Offset di M[i,j]
STRH R0, [R3, R4]    @ Copia R0 in M[i,j]
```

Stack

- Lo **stack** (o pila) è una struttura cosiddetta "LIFO" (Last In First Out), ovvero che mantiene una serie di impilati uno sull' altro
- Le operazioni sono
  - **PUSH**: aggiungere un nuovo dato sopra agli altri già presenti
  - **POP**: togliere dalla pila il dato che sta in cima
- Ad uno stack è associato un puntatore alla testa dello stack (**stack pointer**)
- In genere, ogni programma ha accesso ad uno stack di sistema
  - Lo stack pointer è mantenuto nel **registro SP**

Tipi di stack

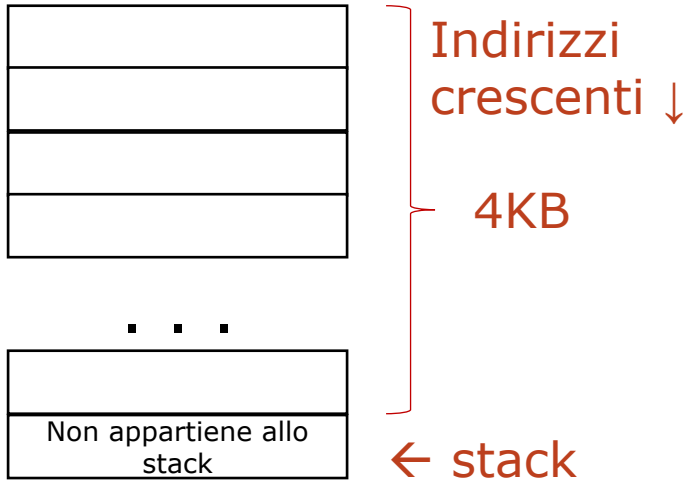
- Possibili varianti di uno stack:
  - Lo stack pointer SP **cresce/decresce** con push
  - SP punta al **primo elemento vuoto/all'ultimo elemento usato**
  - 4 possibili implementazioni

- Nel corso usiamo uno stack **full descending:**
  - SP decresce con push
  - Punta all'ultimo elemento usato

Stack full descending

- Definizione di un'area di memoria di 4KB riservata ad uno stack (full descending):

```
        .equ STL, 4096
      .skip  STL
stack: .skip  4
```



- Inizializzazione dello stack pointer SP:

```
LDR SP, =stack
```

Push e pop in stack full descending

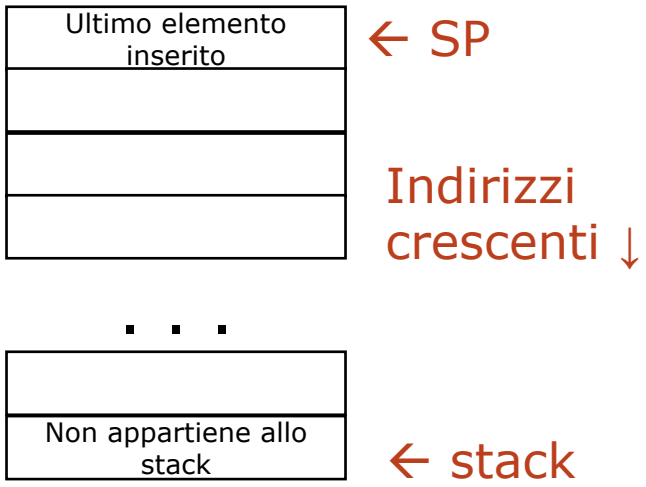
SP = R13: lo stack pointer si trova in R13

Operazione push: PUSH {R0}

- SP = SP – 4
- M[SP] = R0

Operazione pop: POP {R0}

- R0 = M[SP]
- SP = SP + 4



Push e pop multipli

Effettua 8 push dei registri R0-R7

```
PUSH {R0-R7}
```

Essendo uno stack full descending, i registri vengono salvati nell'ordine: R7, R6, …, R0

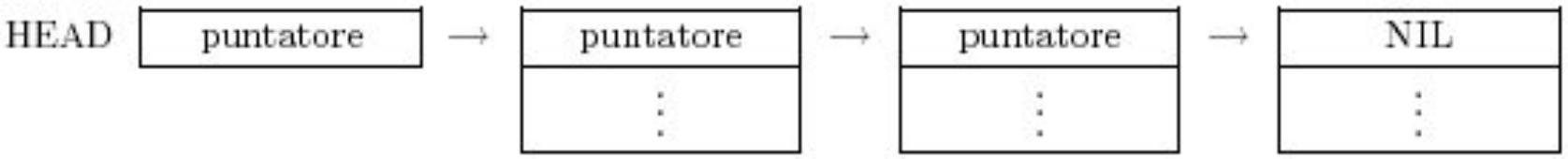
Effettua 8 pop e salva il contenuto nei registri R0-R7

```
POP {R0-R7}
```

Essendo uno stack full descending, i registri vengono salvati nell'ordine: R0, R1, …, R7

Liste concatenate

- Ad ogni elemento di una **lista concatenata (linked list)** è associato un puntatore che individua l'elemento successivo.
- Gli elementi contengono un **campo puntatore** (4 byte) e possono trovarsi ovunque in memoria.
- La lista contiene un puntatore HEAD al primo elemento (o a NIL)



Liste concatenate

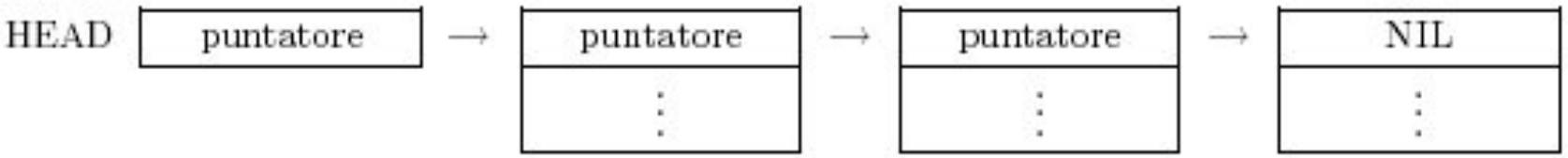
- Inizializzazione di una lista concatenata vuota:

```
.equ NIL, -1
HEAD: .word NIL
```

- Se R1 punta a un elemento, con l'istruzione:

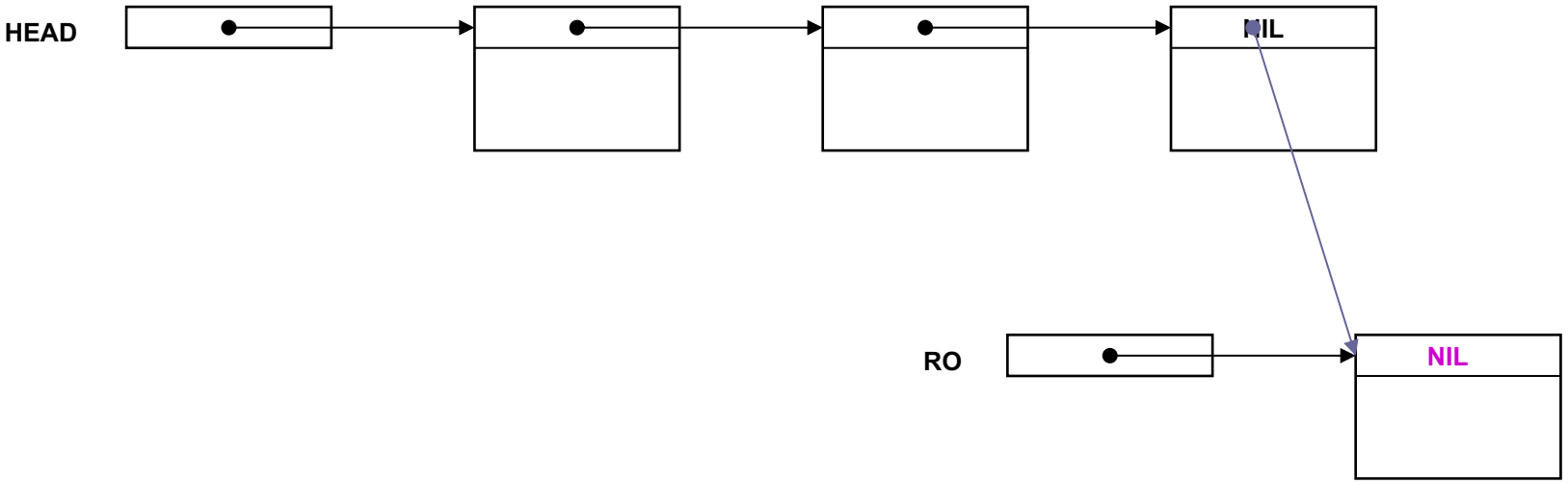
LDR R1, [R1]

R1 punta all'elemento successivo (e si percorre la lista).



Inserzione di un elemento alla fine

L'elemento da aggiungere è puntato da R0



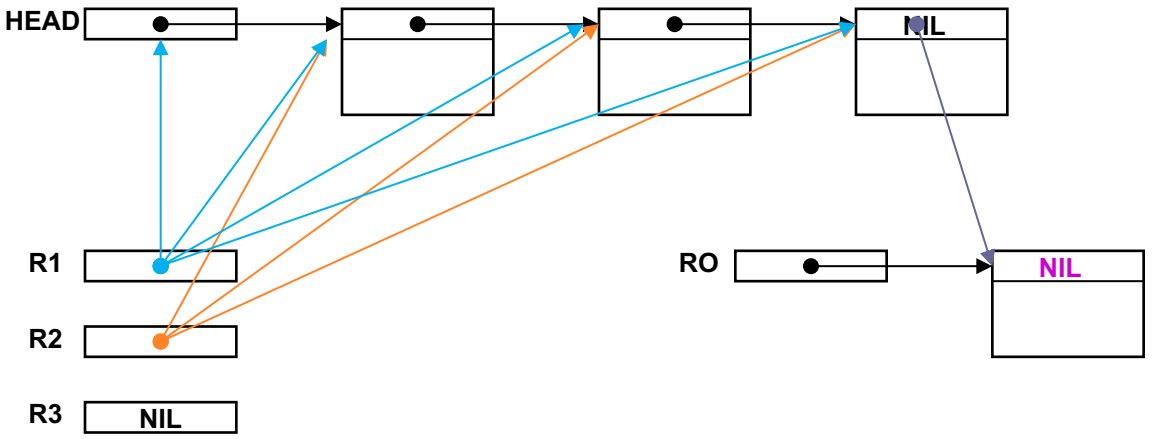
Inserzione: codice

Inserzione, a fine lista, di un elemento puntato da R0

```
       LDR R1, =HEAD   @ Puntatore iniziale
       MOV R3, #NIL    @ Copio NIL in R3
CERCA:LDR R2, [R1]     @ Carico indirizzo in R2
       CMP R2, #NIL    @ E' ultimo elemento (R2=NIL)?
       BEQ ULTIMO      @ Se ultimo salta a ULTIMO
       MOV R1, R2      @ Altrimenti passa al prossimo
       B  CERCA        @ Ripeti
ULTIMO:STR R0, [R1]    @ Aggancio
       STR R3, [R0]    @ Ultimo elemento punta a NIL
```

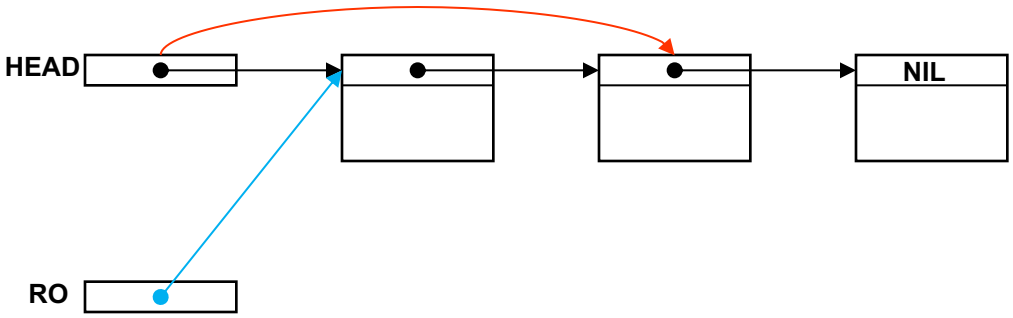
Inserzione: esempio

```
      LDR R1, =HEAD
      MOV R3, #NIL
CERCA:LDR R2, [R1]
      CMP R2, R3
      BEQ ULTIMO
      MOV R1, R2
      B  CERCA
ULTIMO:   STR R0, [R1]
      STR R3, [R0]
```



Estrazione del primo elemento

Rimuovere il primo elemento della lista e inserire un puntatore all'elemento rimosso in R0



Estrazione: codice

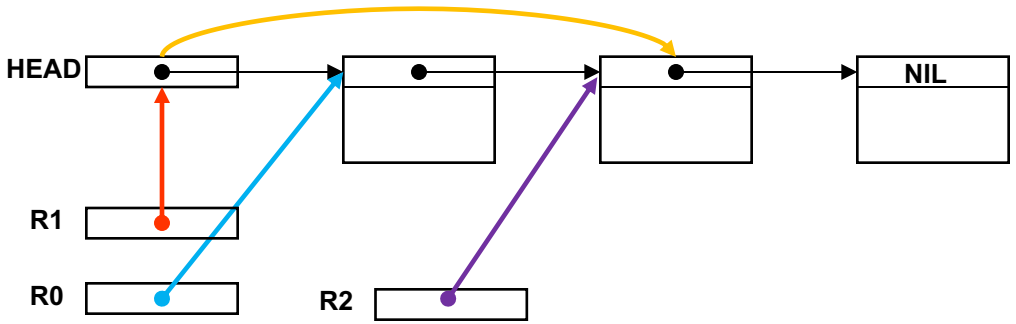
Estrazione del primo elemento (puntatore in R0)

```
LDR R1, =HEAD  @ Puntatore iniziale
LDR R0, [R1]   @ Puntatore al primo elemento
CMP R0, #NIL   @ Lista vuota?
BEQ VUOTA      @ Si, non estrae
LDR R2, [R0]   @ Indirizzo prossimo elemento
STR R2, [R1]   @ Inserisci in HEAD

VUOTA: . . .
```

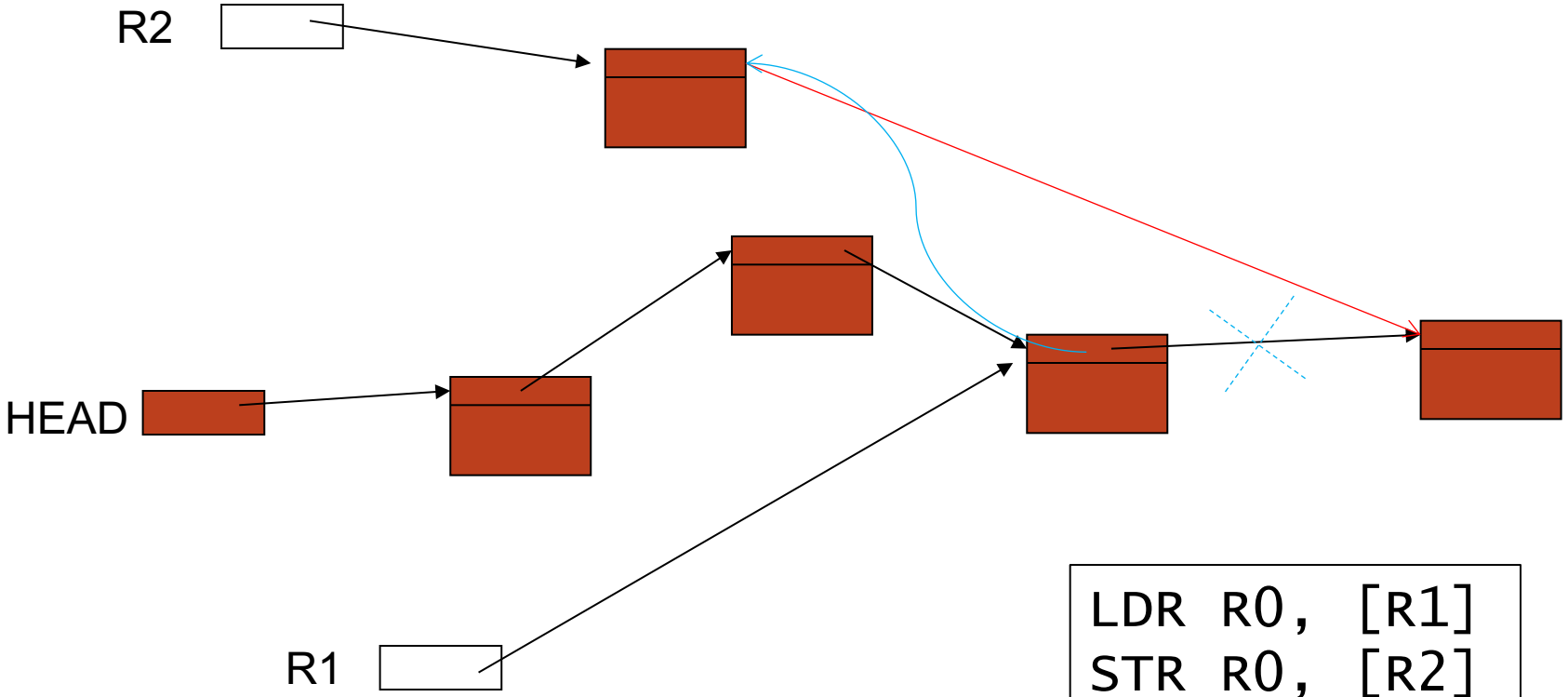
Estrazione: esempio

```
LDR R1, =HEAD
LDR R0, [R1]
CMP R0, #NIL
BEQ VUOTA
LDR R2, [R0]
STR R2, [R1]
```



Inserzione in posizione generica

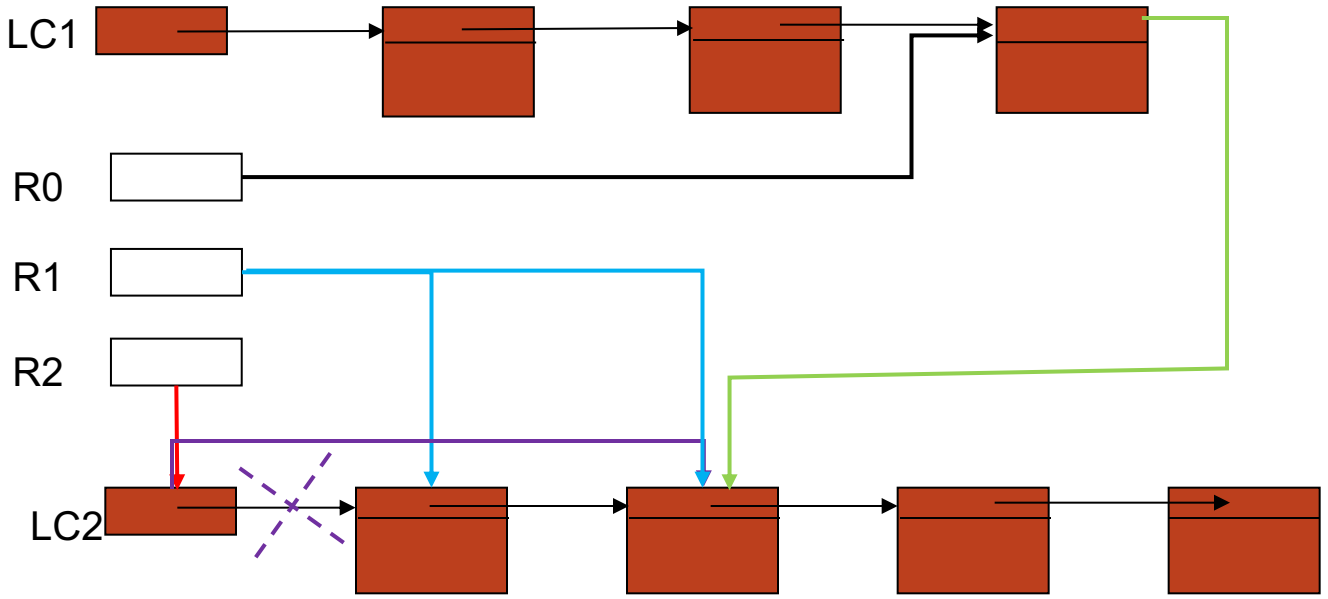
Inserzione dell'elemento puntato da R2 nell posizione successiva a quella dell'elemento puntato da R1.




```
LDR R0, [R1]
STR R0, [R2]
STR R2, [R1]
```

Unione di due liste

LC1 e LC2 sono 2 liste concatenate. In R0 vi sia un puntatore all'ultimo elemento di LC1.
Rimuovere il primo elemento di LC2 e agganciare l'intera lista LC2 appena modificata in coda ad LC1.




```
LDR R2, =LC2
LDR R1, [R2]
LDR R1, [R1]
STR R1, [R2]
STR R1, [R0]
```

Liste concatenate: esercizi proposti

- Gli esempi visti (inserzione alla fine, estrazione dalla testa) corrispondono ad usare la lista con modalità FIFO (coda);
- Si propongono altri esercizi:
  - inserzione e estrazione usando la lista con modalità LIFO (entrambe dalla testa);
  - inserzione di un dato in una lista ordinata (va prima individuato il punto in cui effettuare l'inserzione);
  - ricerca di un dato in una lista ordinata;
  - estrazione di un elemento da una lista ordinata, se presente.