Lezione 17: Indirizzamento della memoria LC-3

Laboratorio di Elementi di Architettura e Sistemi Operativi

22 Maggio 2013

Riassunto sull'architettura LC-3

1. Organizzazione della memoria:
   - La memoria è composta da locazioni (*word*) a 16 bit.
   - Gli indirizzi sono a 16 bit.
   - Gli indirizzi di memoria disponibili per l'utente vanno da 0x3000 a 0xFDFF
2. Insieme dei registri:
   - 8 registri (chiamati R0, R1, ..., R7) a 16 bit (word);
3. Insieme delle istruzioni:
   - *operazioni*: processano informazioni;
   - *spostamento di dati*: spostano informazioni tra registri e memoria e viceversa;
   - *controllo*: cambiano il flusso d'esecuzione delle istruzioni.
4. Tipi di dato:
   - interi in complemento a due
5. Metodi di indirizzamento:
   - Il *metodo di indirizzamento* è il meccanismo per specificare dove si trova un operando;
   - LC-3 supporta cinque modi di indirizzamento:
     - all'interno dell'istruzione stessa (literal o immediate)
     - a registro
     - tre a memoria

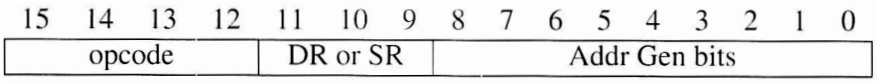
Oggi vedremo i modi di indirizzamento a memoria:

- PC-relative
- indiretto
- base + offset

Le istruzioni di LC-3: spostamento di dati e metodi di indirizzamento

Struttura delle istruzioni di spostamento dati

- LC-3 definisce 7 istruzioni per spostare dati: LD, LDR, LDI, LEA, ST, STR e STI;
- In generale il formato delle istruzioni di load e store è:



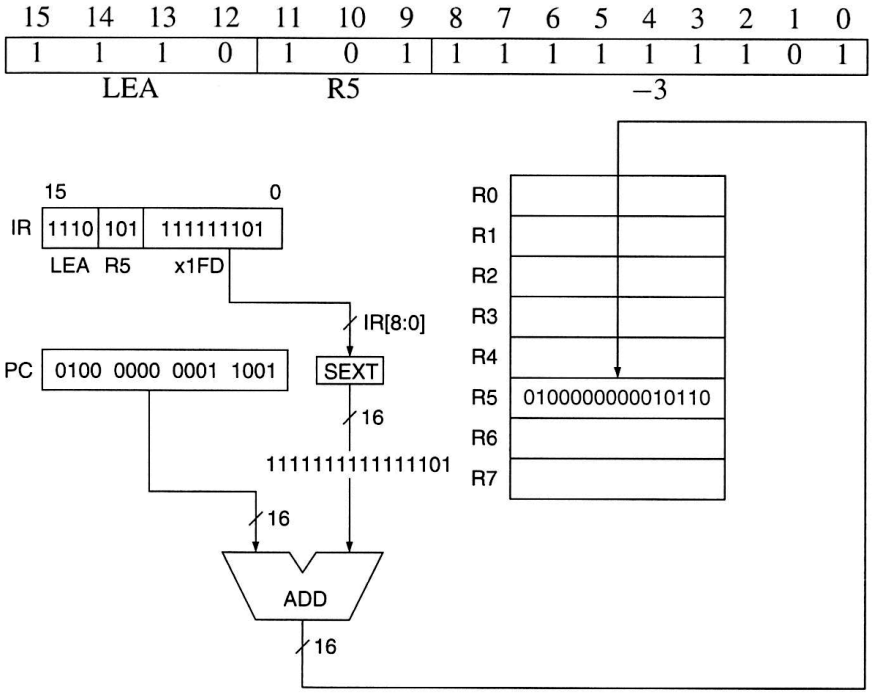
- Il primo operando è un *registro*. Gli altri operandi dipendono dall'istruzione.
- I bit [0:8] contengono *l'address generation bits*, cioè l'informazione usata per calcolare l'indirizzo a 16 bit del dato in memoria.
- **Il *modo di indirizzamento* specifica come devono essere interpretati questi bit**.

Indirizzamento immediato

```
LEA DR,<label>
```

L'istruzione LEA (opcode = 1110) carica nel registro specificato dai bit [9:11] il valore ottenuto sommando il valore del PC incrementato con il valore indicato nei bit [0:8] dell'istruzione (rappresentato su 16 bit).

- Di solito l'istruzione LEA viene usata per inizializzare un registro con un valore molto vicino all'indirizzo di memoria che si sta usando;
- Esempio: supponiamo che la locazione di memoria 0x4018 contenga l'istruzione "LEA R5, #-3" e il PC sia 0x4018; dopo l'esecuzione dell'istruzione 0x4018, R5 conterrà 0x4016.



Indirizzamento PC-relative

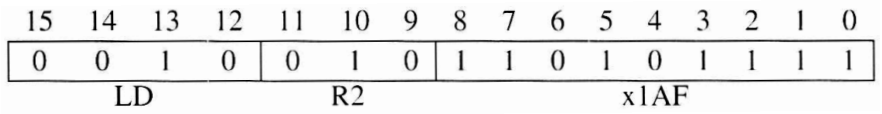
```
LD DR,<label>    ST DR,<label>
```

- Si chiama così perchè i bit [0:8] specificano l'offset da sommare al PC;

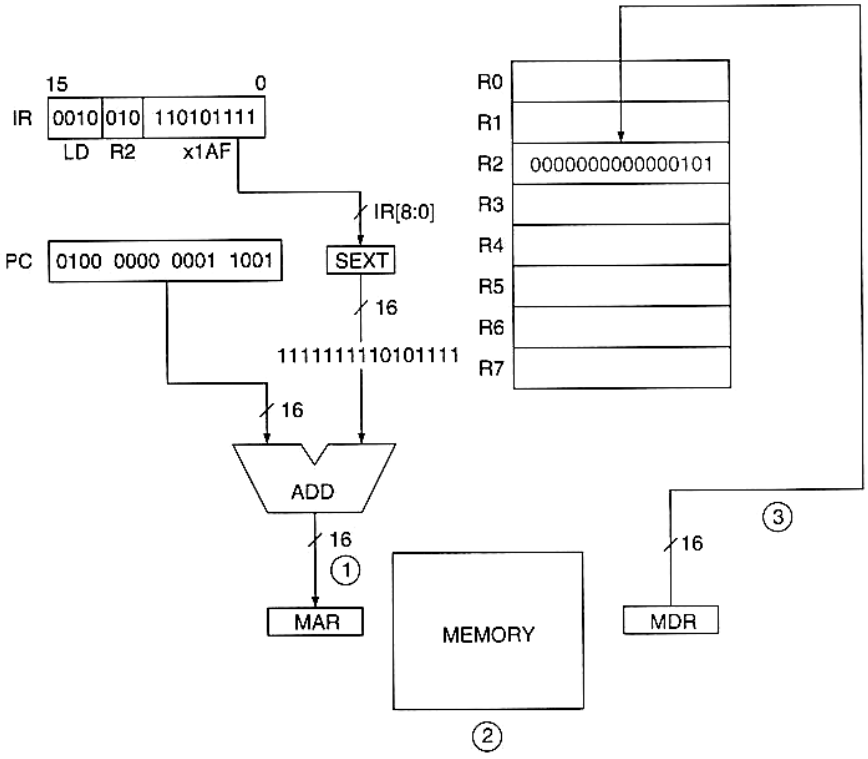
  L'indirizzo di memoria a cui fa riferimento l'operazione viene calcolato sommando al valore del PC incrementato il valore, esteso su 16 bit, indicato dai bit [0:8] dell'istruzione.

- Se l'operazione è di load, viene messo nel registro indicato dai bit [9:11] il valore contenuto all'indirizzo di memoria calcolato; se è di store, viene memorizzato all'indirizzo di memoria calcolato il valore contenuto nel registro indicato dai bit [9:11].

- Esempio: supponiamo che il valore del PC sia 0x4018 e l'istruzione sia "LD R2, x1AF"; in R2 verrà caricato il valore contenuto in memoria all'indirizzo 0x3FC8.



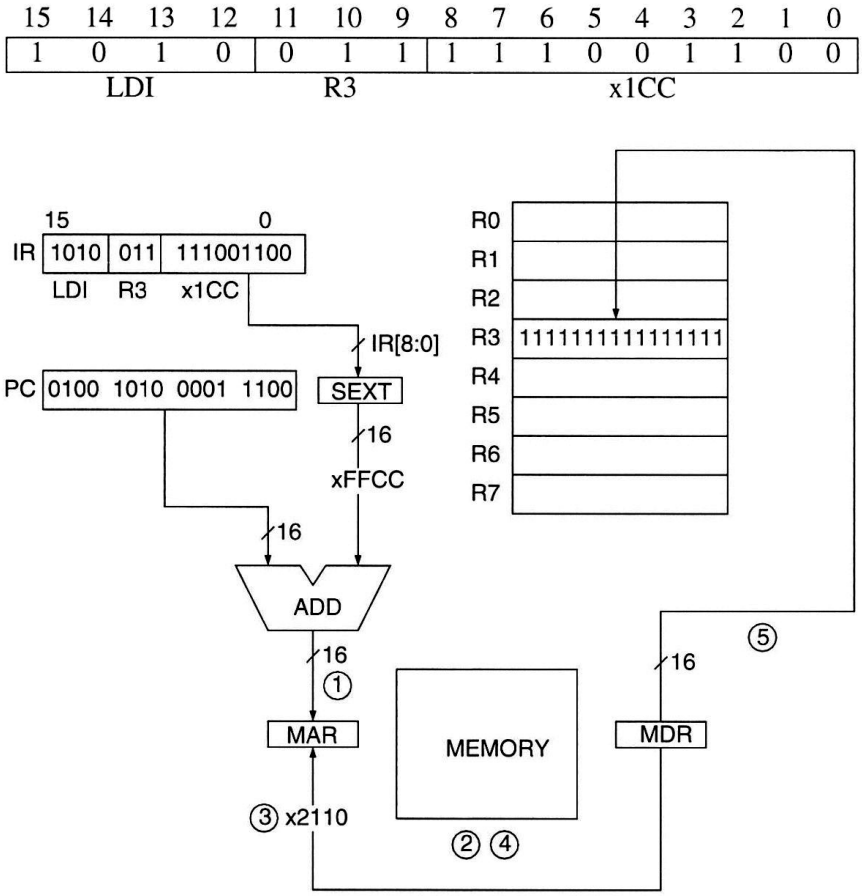
- Per eseguire l'istruzione è necessario effettuare un *accesso alla memoria*.



Indirizzamento indiretto

```
LDI DR,<label>    STI DR,<label>
```

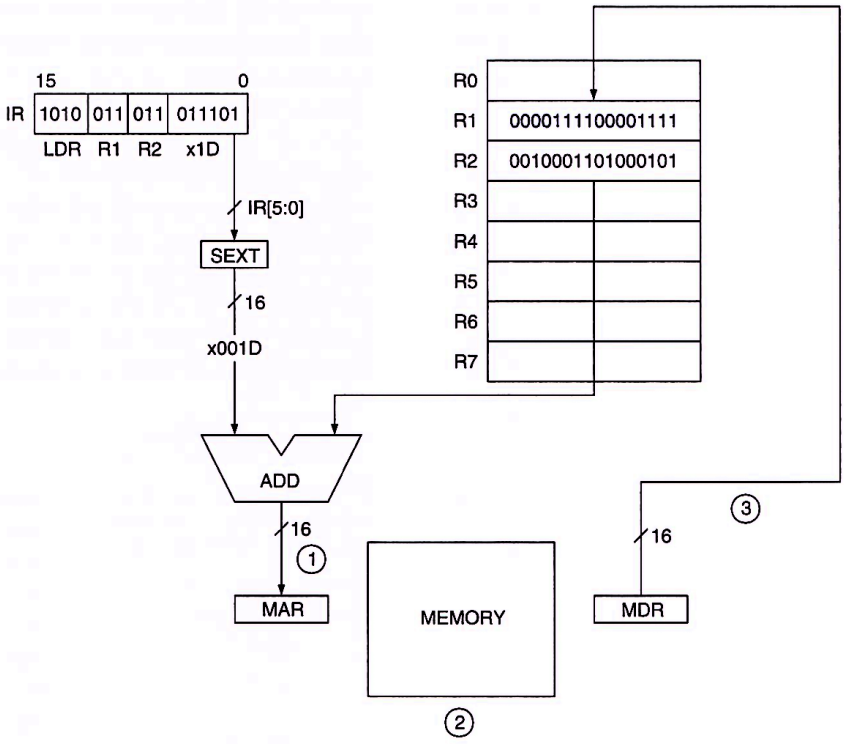
- Le istruzioni **LDI** (opcode = 1010) e **STI** (opcode = 1011) utilizzano l'indirizzamento indiretto.
- In questo caso l'indirizzo calcolato dalla somma tra il PC e il valore specificato nell'istruzione, *non rappresenta direttamente l'indirizzo di memoria da/su cui leggere/scrivere,* ***ma l'indirizzo di memoria che contiene l'indirizzo dell'operando da caricare/salvare***;
- Esempio: supponiamo che l'istruzione in 0x4A1B sia "LDI R3, x1CC", e che la zona di memoria 0x49E8 (ottenuto sommando il PC incrementato, 0x4A1C, all'estensione su 16 bit con segno dell'offset, che da 0xFFCC) contenga 0x2110; l'esecuzione sarà:
  - somma dell'offset al PC (il cui risultato è l'indirizzo 0x49E8 (fase 1);
  - il contenuto della cella di memoria all'indirizzo 0x49E8 (0x2110) rappresenta l'indirizzo di memoria vero e proprio a cui l'operazione fa riferimento (fase 2);
  - tale indirizzo (0x2110) viene caricato nel MAR (fase 3);
  - viene preso il contenuto della cella di memoria 0x2110 e il suo contenuto viene messo nel registro di destinazione (R3) (fasi 4 e 5).



Indirizzamento base+offset

`LDR DR,BR,<offset>` `STR DR,BR,<offset>`

- L'indirizzo dell'operando è ottenuto aggiungendo l'offset indicato dai 6 bit [0:5], esteso a 16 bit, al registro di base indicato dai bit [6:8].
- I 6 bit utilizzati per rappresentare l'offset vanno sempre considerati come interi in complemento a 2 (quindi valori compresi fra -32 e +31);
- Esempio: si supponga che R2 contenga il valore 0x2345, e che l'istruzione da eseguire sia "LDR R1, R2, x1D"; dopo la sua esecuzione, in R1 si avrà il contenuto della cella di memoria calcolata sommando 0x2345 a 0x1D.



Pseudo direttive assembly per la gestione della memoria

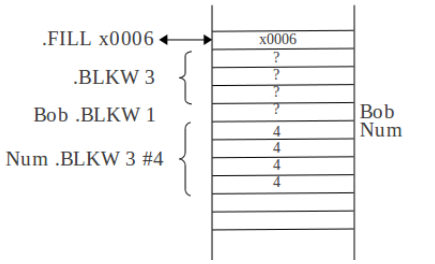
- Per scrivere un programma in assembly sono necessarie alcune *pseudo direttive*.
- Servono in fase di traduzione da ASM a linguaggio macchina per far si che il traduttore interpreti correttamente il contenuto del programma.
- Nella lezione scorsa abbiamo visto le pseudo-direttive **.ORIG** e **.END**, che specificano l'inizio e la fine del codice del programma.

Pseudo direttive assembly per la gestione della memoria

- **.FILL**: indica che la locazione di memoria contiene il valore indicato dall'operando; ad esempio `.FILL x0006` specifica che una certaa locazione contiene il valore `x0006`;

- **.BLKW**: indica di riservare una sequenza (contigua), lunga tanto quanto indicato dall'operando, di locazioni di memoria.

  Si possono anche inizializzare;




```
        .ORIG     x3010
HELLO   .STRINGZ  "Hello, World!"
```

```
x3010: x0048
x3011: x0065
x3012: x006C
x3013: x006C
x3014: x006F
x3015: x002C
x3016: x0020
x3017: x0057
x3018: x006F
x3019: x0072
x301A: x006C
x301B: x0064
x301C: x0021
x301D: x0000
```

- **.STRINGZ**: indica di iniziare una sequenza lunga n+1 di locazioni di memoria. L'argomento è una sequenza di n caratteri compresa fra i doppi apici.

  I caratteri vengono rappresentati su 16bit.

  Il carattere terminatore `'\0'` è sottointeso.

Esempio: somma di un vettore di 12 elementi

Esempio: somma dodici numeri memorizzati a partire da una locazione di memoria. Di seguito è riportata una possibile implementazione in linguaggio C ...

```
int main(){
 int A[12]={1,2,3,4,5,6,7,8,9,10,11,12};
 int i;
 int r=0;
 int *p = &A[0];
 for(i=12;i>0;i--){
   r=r+(*p);
   p++;
 }
}
```

... ed il suo corrispondente in assembly LC-3:

```
.ORIG x3000
      AND R1,R1,#0  ; R1 contiene la somma r
      AND R4,R4,#0  ; R4 e' il contatore i
      ADD R4,R4,#12 ; inizializza R4
      LEA R2,DATA   ; p = &A[0]
LOOP  LDR R3,R2,#0  ; R3 = *p
      ADD R1,R1,R3  ; fai la somma
      ADD R2,R2,#1  ; incrementa p
      ADD R4,R4,#-1 ; decrementa il contatore
      BRp LOOP      ; se positivo torna a LOOP
      ST  R1, SUM   ; salva il risultato
      HALT

DATA  .FILL #1      ; riserva 12 celle
      .FILL #2      ; con i numeri da sommare
      .FILL #3
      .FILL #4
      .FILL #5
      .FILL #6
      .FILL #7
      .FILL #8
      .FILL #9
      .FILL #10
      .FILL #11
      .FILL #12
SUM   .BLKW 1       ; riserva una cella
                    ; per il risultato
.END
```