

# Lezione 2: L'architettura LC-3

## Laboratorio di Elementi di Architettura e Sistemi Operativi

16 Gennaio 2020

### Pseudo direttive assembly

- Per scrivere un programma in assembly sono necessarie alcune pseudo direttive (chiamate anche pseudo-ops);
- esse non sono vere e proprie istruzioni da eseguire, ma...

**servono in fase di traduzione da ASM a linguaggio macchina per far sì che il traduttore interpreti correttamente il contenuto del programma.**

- si riconoscono perché iniziano con il "." e, ovviamente, sono nomi riservati.
- Le pseudo-direttive del linguaggio assembly di LC-3 sono:

**.ORIG:** indica l'indirizzo di partenza del programma LC-3; ad esempio alla riga 05 viene detto che il programma deve iniziare all'indirizzo (esadecimale su 16 bit) x3050;

**.END:** indica dove finisce il programma. Attenzione: questa istruzione non dice che il programma deve essere terminato, ma l'indirizzo a cui terminano le sue istruzioni!

**.FILL:** indica che la prossima locazione di memoria contiene il valore indicato dall'operando;

**.BLKW:** indica di riservare una sequenza (contigua), lunga tanto quanto indicato dall'operando, di locazioni di memoria.  
Si possono anche inizializzare;

**.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.

## ISTRUZIONE Load Effective Address ( LEA ):

- Si usa SOLO con l'istruzione di caricamento effettivo di un indirizzo (Load Effective Address LEA);
- L'istruzione LEA (codice operativo = 1110) carica nel registro specificato nelle posizioni [9:11] il valore ottenuto sommando il valore del PC incrementato con il valore indicato nelle posizioni [0:8] dell'istruzione (rappresentato su 16 cifre binarie).
- Di solito l'istruzione LEA è usata per inizializzare un registro con un indirizzo molto vicino all'indirizzo di memoria che si sta usando;
- **Esempio:** supponiamo che la locazione di memoria 0x4019 contenga l'istruzione "LEA R5, #-3" e il PC sia 0x4019; dopo l'esecuzione dell'istruzione 0x4019, R5 conterrà 0x4016

```
.ORIG x3000
    LEA R0, DODICI
    LD  R1, CINQUE
```

; Definizione di variabili

```
DODICI    .FILL 0012    ; Valore espresso in decimale
CINQUE    .FILL x0005    ; Valore espresso in esadecimale
```

```
.END
```

## ISTRUZIONE LOAD ( LD ):

- Le istruzioni **LD** (codice operativo = 0010) e **ST** (codice operativo = 0011) specificano l'indirizzamento relativo al contatore di programma (PC-relative);
- Si chiama così perchè le posizioni [0:8] specificano lo scostamento da sommare al contatore di programma (PC);
- L'indirizzo di memoria a cui fa riferimento l'operazione è calcolato sommando al valore del PC incrementato il valore, esteso su 16 cifre binarie, indicato dalle posizioni [0:8] dell'istruzione.
- Se l'operazione è di load, si scrive nel registro indicato dalle posizioni [9:11] il valore contenuto all'indirizzo di memoria calcolato; se è di store, si memorizza all'indirizzo di memoria calcolato il valore contenuto nel registro indicato dalle posizioni [9:11].

### Esempio:

Scrivere il seguente programma, compilarlo e mandarlo in esecuzione

```
.ORIG x3000
    LD R0, DODICI
    LD R1, 0          ; Verificare il valore del program counter e controllare cosa viene inserito in R1
    LD R2, CINQUE

; Definizione di variabili
DODICI    .FILL 0012  ; Valore espresso in decimale
CINQUE    .FILL x0005 ; Valore espresso in esadecimale

.END
```

## ISTRUZIONE LOAD ( LDI ):

- Le istruzioni **LDI** (codice operativo = 1010) e **STI** (codice operativo = 1011) specificano l'indirizzamento indiretto;
- Funzionano esattamente come le istruzioni LD e ST, ma l'indirizzo calcolato dalla somma tra il PC e il valore specificato nell'istruzione non è l'indirizzo della parola di memoria da/in cui leggere / scrivere l'operando, **ma è l'indirizzo della parola di memoria che contiene l'indirizzo della parola di memoria da/in cui leggere/scrivere l'operando;**

### Esempio:

Scrivere il seguente programma, compilarlo e mandarlo in esecuzione

```
.ORIG x3000
    LDI R0, PLACE
    LD R2, CINQUE

    HALT

; Definizione di variabili
DODICI    .FILL 0012    ; Valore espresso in decimale
CINQUE    .FILL x0005   ; Valore espresso in esadecimale

PLACE     .FILL X45A7   ; Andare alla locazione X45A7 e inserire un valore

.END
```

## ISTRUZIONE LOAD ( LDR ):

- Le istruzioni **LDR** (codice operativo = 0110) e **STR** (codice operativo = 0111) specificano l'indirizzamento base+scostamento;
- L'indirizzo dell'operando è ottenuto aggiungendo lo scostamento indicato dalle 6 cifre binarie [0:5], esteso a 16 cifre binarie, al registro di base indicato dalle cifre binarie [6:8].
- le 6 cifre binarie utilizzate per rappresentare lo scostamento vanno sempre considerate 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 0x0F0F della cella di memoria il cui indirizzo 0x2362 è stato calcolato sommando 0x2345 a 0x1D.

### Esempio:

.ORIG x3000

LEA R1, DODICI ; R1 = Indirizzo di DODICI  
LDR R0, R1, 0 ; Puo' essere comodo per la gestione dei vettori

LD R2, CINQUE

HALT

; Definizione di variabili

DODICI .FILL 0012 ; Valore espresso in decimale

CINQUE .FILL x0005 ; Valore espresso in esadecimale

.END

Tradurre in assembler le seguenti istruzioni e provarle sul ISS LC3:

### Esercizio 1:

```
int a = 25;
int b = 0;
b = a;
a = a + 40;

LD R0, A
ST R0, B

LD R1, QUA
ADD R0, R1, R0
ST R0, A

HALT

QUA .FILL 40
A .FILL 25
B .FILL 0
.END
```

### Esercizio 2:

```
int base = 14;
int altezza = 40;
int doppiabase;
int differenza;

doppiabase = base + base;
differenza = altezza - doppiabase;
```

### Esercizio 3:

```
int basemaggiore = 43;
int baseminore = 30;
int diagonale = 10;
int perimetro = 0;

perimetro = basemaggiore + baseminore + diagonale + diagonale;
```

## ISTRUZIONE BRANCH ( BR ):

- L'istruzione di branch è **BR** (codice operativo = 0000);
- Il suo formato è:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	x	y	x	p	p	p	p	p	p	p	p	p
BR				N	Z	P	PC offset								

- Le cifre binarie [9:11] corrispondono alle tre condizioni N, Z e P;
- esse sono usati per cambiare il flusso di esecuzione;
- Durante la fase di ESECUZIONE il processore esamina le cifre N, Z, P, e:
  - se la cifra binaria [11] = 1, si esamina la condizione corrispondente a N;
  - se la cifra binaria [10] = 1, si esamina la condizione corrispondente a Z;
  - se la cifra binaria [9] = 1, si esamina la condizione corrispondente a P;

### Esempio:

.ORIG x3000

```
AND R0, R0, 0      ; R0 <- 0
AND R1, R1, 0      ; R1 <- 0
BRz INCREMENTAR1  ; Salta se è settato il flag zero
ADD R0,R0, 1
```

INCREMENTAR1:

```
ADD R1, R1, 1
```

```
HALT
```

; Definizione di variabili

.END

Esempio il ciclo FOR:

convertire in assembler il seguente programma C

```
int vet[10];
int i;
for (i = 0; i < 10; i++)
    v[ i ] = i;
```

.ORIG x3000

LEA R0, vet

AND R1, R1, 0 ; utilizzo R1 come variabile i, la devo azzerare

iniziofor:

ADD R2, R1, -10

BRz finefor

STR R1, R0, 0

ADD R0, R0, 1 ; vado alla cella successiva del vettore

ADD R1, R1, 1 ; incremento i

BRnzp iniziofor

finefor:

HALT

; Definizione di variabili

vet .BLKW 0010 ; Valore espresso in decimale

.END

Esempio il ciclo WHILE:

convertire in assembler il seguente programma C

```
int A[20]={1,2,3,4,5,6,7,8,9,10,11,12,-1,0,0,0,0,0,0,0};
int i=0;
int r=0;
while(A[i]>0){ //Nota: -1 è la sentinella
    r=r+A[i];
    i++;
}
```

.ORIG x3000

LEA R1, A

AND R3, R3, 0 ;R3 come variabile r, la devo azzerare

iniziowhile:

LDR R4, R1, 0 ;R4 <- A[i]

BRnz finewhile

ADD R3, R3, R4 ;R3 <- R3 + A[i]

ADD R1, R1, 1 ; incremento i

BRnzp iniziowhile

finewhile:

HALT

; Definizione di variabili

A .FILL 0001 ; Valore espresso in decimale

.FILL 0002

.FILL 0003

.FILL 0004

.FILL 0005

.FILL 0006

.FILL 0007

.FILL 0008

.FILL 0009

.FILL 0010

.FILL 0011

.FILL 0012

.FILL 0013

.FILL 0014

.FILL -1

.FILL 0

.FILL 0

.FILL 0

.FILL 0

.FILL 0

.FILL 0

.END

Esercizi si traducano in assembler LC3 i seguenti spezzoni di codice:

1. int A = 10;  
int B = 15;  
A = A + B;

2. int A = 20;  
int B = 7;  
A = A - B;

3. int A = 5;  
A = A \* 7;

4. int A = 13;  
A = A / 3;

5. long A = 0xFFF; ( long = 32 bit)  
long B = A + 5;

# SOLUZIONI

## Esercizio1:

```
.ORIG x3000
```

```
LD R0, A  
ST R0, B
```

```
ADD R0, R0, #10  
ST R0, A
```

```
HALT
```

; Definizione di variabili

A .FILL 0025 ; Valore espresso in decimale

B .FILL x0000 ; Valore espresso in esadecimale

```
.END
```

```
1. int A = 10;  
   int B = 15;  
   A = A + B;
```

## Esercizio1:

```
.ORIG x3000
```

```
LD R0, A  
LD R1, B
```

```
ADD R0, R1, R0  
ST R0, A
```

```
HALT
```

; Definizione di variabili

A .FILL 0010 ; Valore espresso in decimale

B .FILL 0015 ; Valore espresso in esadecimale

```
.END
```

```
2. int A = 20;
   int B = 7;
   A = A - B;
```

### **Esercizio2:**

```
.ORIG x3000
```

```
LD R0, A
LD R1, B
NOT R1
ADD R1, R1, 1
```

```
ADD R0, R1, R0
ST R0, A
```

```
HALT
```

```
; Definizione di variabili
```

```
A .FILL 0020 ; Valore espresso in decimale
```

```
B .FILL 0007 ; Valore espresso in esadecimale
```

```
.END
```

```
3. int A = 5;  
   A = A * 7;
```

```
.ORIG x3000
```

```
LD R0, A  
LD R2, A  
AND R1, R1, 0  
ADD R1, R1, -7
```

```
inizio:
```

```
ADD R0, R0, R2
```

```
ADD R1, R1, 1
```

```
BRzp fine:
```

```
BRznp inizio
```

```
fine:
```

```
ST R0, A
```

```
HALT
```

```
; Definizione di variabili
```

```
A .FILL 0005 ; Valore espresso in decimale
```

```
.END
```

```
4. int A = 13;  
   A = A / 3;
```

```
5. long A = x0FFF; ( long = 32 bit)  
   long B = A + 5;
```