Lezione 7: Il linguaggio assembly di LC-3

Elementi di Architettura e Sistemi Operativi
Docenti: Tiziano Villa, Davide Bresolin

Corso di Laurea in Bioinformatica

Dicembre 2013

Abbiamo visto...

- L'architettura delle istruzioni di LC-3;
- Ogni istruzione è identificata da una parola binaria;
- Ogni locazione di memoria è individuata da un indirizzo a 16 bit.

Tuttavia...

scrivere programmi in formato binario non è certo proponibile in pratica. Vediamo quindi come rappresentare in modo più comprensibile i nostri programmi.

Ricorda...

- un algoritmo descrive con un linguaggio "naturale" un insieme di passi da fare per risolvere un problema;
- l'algoritmo viene trasformato in programma attraverso l'uso di un linguaggio di programmazione che può essere:
  - ad alto livello: C, JAVA, Pascal, COBOL, ...;
  - ad un livello più vicino al linguaggio macchina: ASSEMBLY (ASM);
  - in linguaggio macchina (specifico per un'architettura).
- nei primi due casi il programma deve essere tradotto in un insieme di istruzioni specifiche per l'architettura prescelta.

Il linguaggio ASSEMBLY

Il linguaggio assembly di LC-3

- è un linguaggio a basso livello;
- ha lo scopo di rendere la programmazione più semplice rimanendo comunque vicino all'architettura;
- vi è una corrispondenza uno a uno tra istruzioni ASM e istruzioni specifiche dell'architettura;
- mette a disposizione nomi simbolici al posto dei codici operativi delle istruzioni; ad esempio ADD identifica il codice operativo 0001;
- permette inoltre di definire nomi simbolici per le locazioni di memoria (symbolic address).
- Vediamo i dettagli del linguaggio ASM di LC-3 attraverso un esempio.

Il linguaggio assembly di LC-3

- Consideriamo il seguente programma scritto in C (quindi ad alto livello) che moltiplica per 6 l'intero memorizzato in NUMBER.
- Nota: abbiamo visto in precedenza che nell'architettura non c'è la moltiplicazione; occorre quindi creare un ciclo (for) che somma NUMBER a se stesso il numero di volte desiderate.

```
int main(){
 int NUMBER; // da inizializzare col valore desiderato
 int r=0;
 for(int i=6;i>0;i--){
   r=r+NUMBER;
 }
}
```

Il linguaggio assembly di LC-3

- Il suo corrispondente in assembly è:

```
01  ;
02  ; Program to multiply an integer by the constant 6.
03  ; Before execution, an integer must be stored in NUMBER.
04  ;
05          .ORIG   x3050
06          LD      R1,SIX
07          LD      R2,NUMBER
08          AND     R3,R3,#0        ; Clear R3. It will
09                                  ; contain the product.
0A  ; The inner loop
0B  ;
0C  AGAIN   ADD     R3,R3,R2
0D          ADD     R1,R1,#-1       ; R1 keeps track of
0E          BRp     AGAIN           ; the iterations
0F  ;
10          HALT
11  ;
12  NUMBER  .BLKW   1
13  SIX     .FILL   x0006
14  ;
15          .END
```

Le istruzioni

LC-3 - istruzioni ASM

- Il formato generale di un'istruzione assembly è:

```
LABEL    OPCODE    OPERANDS    ; COMMENTS
```

- LABEL (etichetta): assegna un nome simbolico ad un indirizzo (che può contenere un'istruzione o un indirizzo di memoria). È opzionale;
- OPCODE (codice operativo) e OPERANDS (operandi): sono specifici per ogni istruzione;
- COMMENTS (commenti): identifica un commento (come il comando // del C).

LC-3 - istruzioni ASM (LABEL)

- Un'etichetta è un nome simbolico che può essere usato per identificare una zona di memoria;
- viene usata nel programma per fare un riferimento esplicito alla zona di memoria associata;
- in LC-3 un'etichetta può essere lunga al massimo 20 caratteri.
- Esempio: la locazione *AGAIN* identifica l'inizio del ciclo alla riga 0C, ed è usata alla riga 0E dove:
  - se il risultato dell'istruzione "ADD R1,R1,#-1" è positivo il programma "salta" alla locazione indicata da AGAIN ed esegue un'altra iterazione;
  - altrimenti si esce dal ciclo.

LC-3 - istruzioni ASM (CODICE e OPERANDI)

- Come già visto, un'istruzione ha un *CODICE OPERATIVO* e un certo numero di *OPERANDI*;
- il CODICE OPERATIVO:
  - è un nome simbolico che corrisponde ad un'istruzione LC-3;
  - l'idea è che i nomi simbolici (ADD, AND, LDR) sono più mnemonici rispetto a 0001, 0101, 0110;
- gli OPERANDI: sono specifici per ogni istruzione; ad esempio:
  - LD R2, NUMBER carica il contenuto della zona di memoria indicato da NUMBER in R2;
  - AND R3,R3,#0 cancella il contenuto di R3;
  - Nota: nelle operazioni che richiedono letterali, come ad esempio l'istruzione appena vista, si utilizza *#* per indicare un decimale (riga 08), *x* per indicare un esadecimale (riga 13), e *b* per indicare un numero binario.

LC-3 - istruzioni ASM (COMMENTI)

- Nel linguaggio assembly i commenti sono indicati dal **";"** ;
- contengono messaggi in linguaggio naturale che non hanno effetti sul processo di esecuzione;
- ciò che segue il *;* viene completamente ignorato (ad esempio la riga 08).

Pseudo direttive assembly

LC-3 - pseudo direttive ASM

- 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 lutare il traduttore a interpretare correttamente il contenuto del programma;

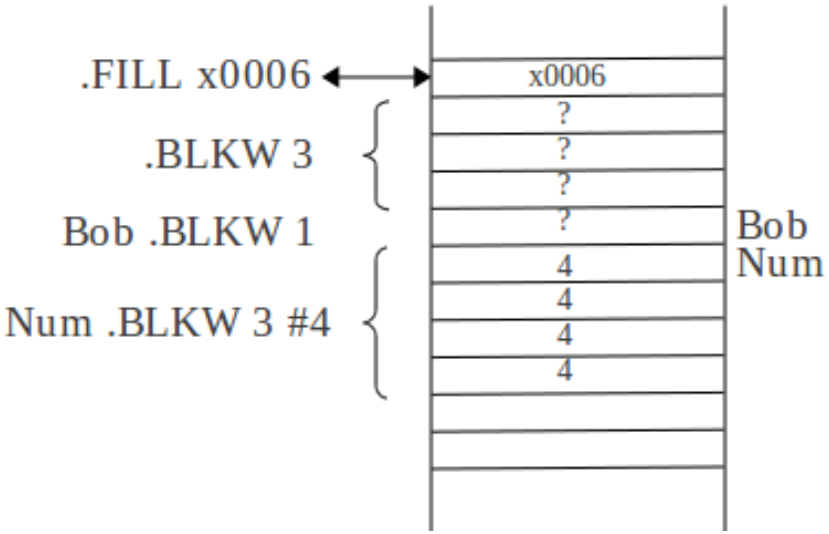
- si riconoscono perchè iniziano con il **"."** e, ovviamente, sono nomi riservati.

LC-3 - pseudo direttive ASM

- Le pseudo-ops del linguaggio ASM 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!

LC-3 - pseudo direttive ASM

- *(cont):*
  - **.FILL**: indica che la prossima locazione di memoria contiene il valore indicato dall'operando; ad esempio alla riga 13 viene detto che la nona locazione (a partire dall'inizio del programma) contiene il valore x0006;
  - **.BLKW**: indica che si riserva una sequenza (contigua) di locazioni di memoria, lunga tanto quanto indicato dall'operando; eventualmente si possono anche inizializzare;



LC-3 - pseudo direttive ASM

- *(cont):*
  - **.STRINGZ**: indica che inizia 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 usando il codice ASCII non esteso. Il carattere terminatore (a cui corrisponde il codice x0000) è sottointeso. Esempio:

```
        .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
```

Da codice assembly a codice macchina

LC-3 - processo di traduzione (1)

- Abbiamo visto che il linguaggio assembly permette di scrivere programmi a più alto livello rispetto alle istruzioni macchina;
- tuttavia un'architettura è in grado di eseguire solo istruzioni in linguaggio macchina: *occorre quindi una traduzione*;
- questo processo di traduzione viene fatto dall'**ASSEMBLATORE (ASSEMBLER)**.

LC-3 - processo di traduzione (2)

- In generale, nel linguaggio assembly vi è una corrispondenza 1 a 1 fra le istruzioni ASM e le istruzioni macchina;
- si può quindi pensare di prendere il programma scritto in ASM e, riga per riga, determinare la corrispondente istruzione macchina.
- Esempio:

```
01  .ORIG x3050
02  AND R3,R3,#0 ; reset di R2
03  LD R1, SIX
...
```

```
01  x3050:    0101011011100000
...
```

- *PROBLEMA: cosa succede quando si arriva alla riga 03?*

LC-3 - processo di traduzione (3)

- **Alla riga 03 l'assemblatore NON SA ANCORA A CHE COSA SI RIFERISCE "SIX"!!!**

Soluzione:

Occorre un processo di traduzione in due fasi:

1. creare la tabella dei simboli;
2. generare il programma in linguaggio macchina.

LC-3 - processo di traduzione (4)

- Creazione della tabella dei simboli:
  - La tabella dei simboli contiene la corrispondenza fra i nomi simbolici (ad esempio AGAIN) e gli indirizzi di memoria a 16 bit;
  - si scorre l'intero programma fra le istruzioni ".ORIG" e ".END" tenendo traccia dell'indirizzo delle istruzioni, e inserendo nella tabella dei simboli le etichette ed il relativo indirizzo;
  - per far questo si usa una locazione chiamata Contatore di Locazione (Location Counter, LC), inizializzato al valore di ".ORIG".
  - Esempio:

SIMBOLO	INDIRIZZO
AGAIN	x3053
NUMBER	x3056
SIX	x3057

LC-3 - processo di traduzione (5)

- Generazione del programma in linguaggio macchina:
  - Si rianalizza riga per riga il programma assembly e con l'ausilio della tabella dei simboli si traduce ogni istruzione nel linguaggio macchina di LC-3;
  - si utilizza ancora una volta LC per determinare gli indirizzi;
  - quando si raggiunge ".END" il processo termina, e il risultato è un file binario corrispondente al programma scritto.

LC-3 - processo di traduzione (6)

La traduzione va fatta a mano?

No! Esistono strumenti che ci aiutano a fare questa traduzione. Ad esempio vedremo *lc3as*, il quale a partire da un file ".asm" genera due file: un file oggetto *.obj* contenente il programma in linguaggio macchina, e un file *.sym* contenente la tabella dei simboli generata dopo la prima analisi del file sorgente.

Come collaudare il nostro programma?

In realta' non esiste una macchina fisica LC-3, ma esiste un simulatore che simula l'architettura di LC-3 sfruttando l'architettura delle macchine x86 o x64. Noi useremo la versione per linux chiamata *lc3sim-tk*.

Esercitazione

LC-3 - Esercitazione

Esempio:

Vogliamo realizzare un programma che sommi tra loro i dieci numeri memorizzati in memoria dalla locazione x3100 alla x3109, e ponga il risultato nel registro R1.

- Quello che dobbiamo fare è:
  - creare il nostro programma assembly, che chiameremo "addnums.asm";
  - memorizzare i dati in memoria.

LC-3 - Esercitazione (addnums.asm)

- Il primo passo è quello di creare il nostro programma assembly;
- utilizzando un qualsiasi editor di testo, creiamo il file "addnums.asm" e scriviamo il nostro programma.
- Una possibile soluzione è:

```
.ORIG x3000
AND R1,R1,x0 ; cancella R1 usato per memorizzare la somma
AND R4,R4,x0 ; cancella R4 usato come contatore
ADD R4,R4,xA ; carica in R4 il valore 10 (contatore)
LEA R2,x0FC ; mette in R2 l'indirizzo di partenza dei dati
LOOP  LDR R3,R2,x0 ; carica il numero da aggiungere
ADD R2,R2,x1 ; incrementa il puntatore
ADD R1,R1,R3 ; aggiunge il numero corrente alla somma
ADD R4,R4,x-1 ; decrementa il contatore
BRp LOOP ; se il contatore non è zero torna a LOOP
HALT
.END
```

LC-3 - Esercitazione (addnums.asm)

- Il prossimo passo consiste nell'invocare l'assemblatore per tradurre il nostro programma assembly in programma macchina.
- Utilizziamo il comando:

```
lc3as addnums.asm
```

- Otteniamo così due file:
  - il nostro programma oggetto: addnums.obj;
  - la tabella dei simboli: addnums.sym;

```
// Symbol table
// Scope level 0:
// Symbol Name       Page Address
// ----------------  ------------
// LOOP              3004
```

LC-3 - Esercitazione (dati.hex)

- Il prossimo passo da fare è mettere i dati in memoria.
- Ci sono due modi:
  - a mano, cliccando sulla cella di memoria e inserendo il valore nel campo "Value"; SCOMODO.
  - attraverso un file di testo che verrà passato al simulatore.
- Noi usiamo il secondo metodo. Creiamo un file chiamato *"data.hex"* dove scriviamo i 10 numeri desiderati. Nota: la prima riga indica al simulatore l'indirizzo di partenza. Esempio:

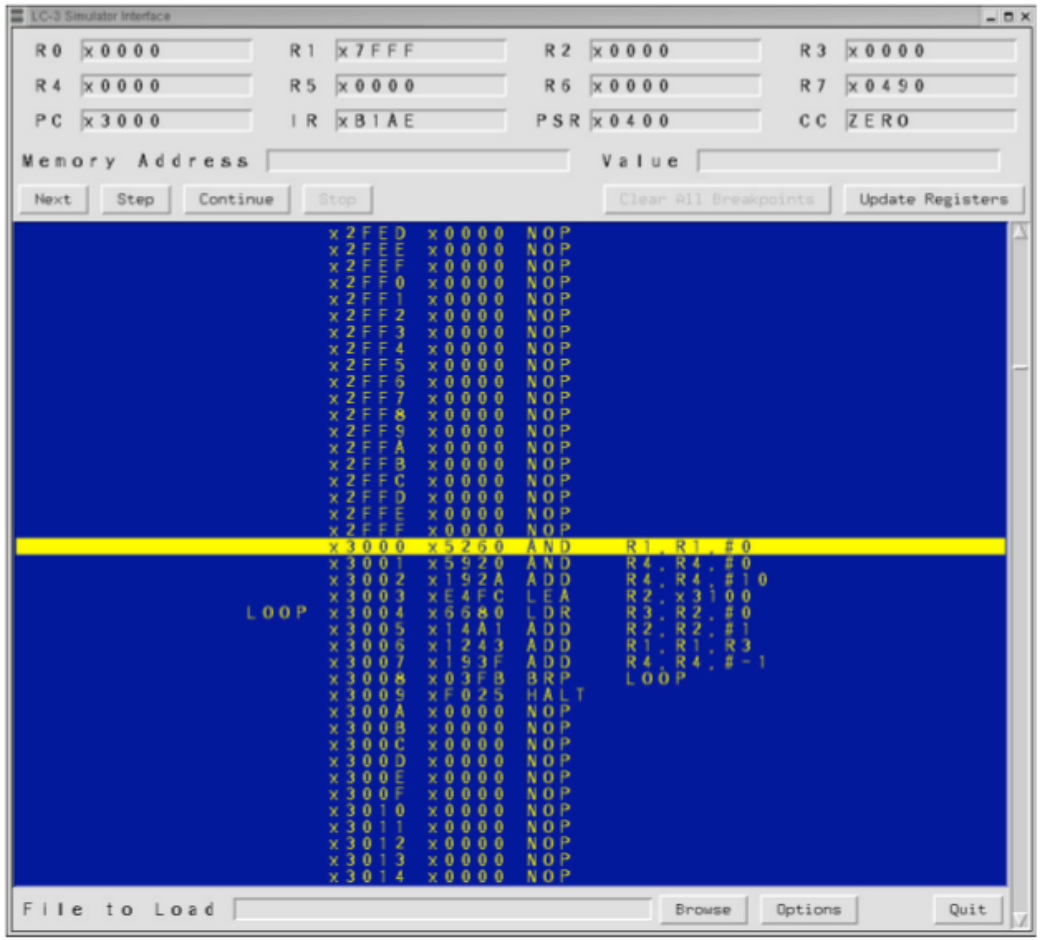
LC-3 - Esercitazione (dati.hex)

```
3100 ; indica che i dati partono dall'indirizzo x3100
x3107 ; numeri da sommare fra loro
x2819
x0110
x0310
x0110
x1110
x11B1
x0019
x0007
x0004
```

- Ovviamente anche questo file deve essere convertito in un file oggetto comprensibile al simulatore;
- Per far questo possiamo usare il comando: "lc3convert -b16 data.hex".

LC-3 - Esercitazione (simulatore)

- A questo punto possiamo lanciare il simulatore con il comando **lc3sim-tk**.



LC-3 - Esercitazione (simulatore)

- Il simulatore è composto da 4 zone principali:
  - in alto troviamo gli 8 registri generali, i registri PC, IR, e CC (condition codes);
  - segue un'area in cui possiamo visualizzare/assegnare gli indirizzi di memoria;
  - segue l'area blu che visualizza tutti gli indirizzi della macchina simulata. In tutto sono 2^{16}=65536 zone di memoria; per visualizzare un indirizzo specifico basta scriverlo nel campo "Memory Address" e dare invio.
  - in basso abbiamo il campo "File to Load" che ci permette di caricare dei file esterni.

LC-3 - Esercitazione (simulatore)

- Possiamo quindi caricare il nostro programma cliccando su ″Browse", e notiamo che esso viene caricato a partire dall'indirizzo x3000.
- Per visualizzarlo usiamo il campo "Memory Address".
- Carichiamo anche il file data.obj che contiene i dati della memoria (che possiamo visualizzare sempre tramite "Memory Address").
- Dopo esserci assicurati che il PC contenga il valore x3000 possiamo eseguire il nostro programma passo passo o per intero.
- Alla fine troveremo la somma calcolata nel registro R1.