Architettura degli Elaboratori
L'Assembler dell'architettura HACK

Andrea Malvezzi

20 novembre, 2024

Contents

1 Che cos'è l'Assembler

L'assembler si occupa di tradurre da linguaggio Assembly a binario il nostro codice. Si passa quindi da istruzioni, etichette e simboli speciali a sequenze di 16 bit, generando in output un file *.hack*.
Oltre a quanto citato, l'assembler si occupa anche di tradurre spazi, righe vuote e simboli come $\backslash n$ o $\backslash t$ (ignorando quindi i commenti).

2 Traduzione di A-instructions

Ricordando la traduzione delle A-instructions spiegata precedentemente, basta porre il primo bit della sequenza a 0 e poi tradurre il numero specificato nel codice. Ad esempio:

$$@3 \Rightarrow 0000000000000011$$

3 Traduzione di C-instructions

Non occorre imparare a memoria la tabella, ma bensì conoscere a grandi linee il significato di una traduzione in bit, qualora se ne trovasse una. Ad esempio:

$$D = A \Rightarrow 1110011000010000$$

Dove le cifre in rosso sono valori fissi delle C-instructions.
Quelli in blue sono le istruzioni inerenti alla *comp*, ovvero l'istruzione effettiva da eseguire (in questo caso A), con davanti il valore di a (qui è 0).
Quella gialla sono la parte *comp*, ovvero la destinazione dove salvare il risultato dell'operazione (nel nostro caso D).
La parte verde è la sezione per il *jump*. Ovviamente, non avendo un jump, ogni bit sarà posto a 0.

4 Gestione dei simboli

I simboli si usano per identificare certi indirizzi nella ROM (destinazione salti) o nella RAM (dove son contenute le variabili).
Alcuni simboli potrebbero essere le lettere, le cifre o caratteri speciali come _,-,. etc ... L'unica accortezza consiste nel non scrivere una cifra all'inizio

di un simbolo (ed ecco spiegato perché non si può cominciare il nome di una variabile con un numero).

4.1 Simboli predefiniti

Alcuni simboli sono pre-esistenti nel linguaggio, ovvero ...

- ... I registri virtuali: I simboli da R0 a R15 fanno riferimento agli indirizzi RAM da 0 a 15;
- ... I puntatori per I/O: I simboli SCREEN e KBD fanno riferimento rispettivamente agli indirizzi RAM 16384 e 24576;
- ... I puntatori di controllo della VM: I simboli SP, LCL, ARG, THIS e THAT fanno riferimento agli indirizzi RAM da 0 a 4, rispettivamente.

4.2 Simboli NON predefiniti

Ci sono poi anche simboli non definiti dal linguaggi, ovvero le etichette e le variabili.
Le prime sono usate per identificare la destinazione dei JUMP e a queste è assegnato come valore l'indirizzo della ROM in cui verrà caricata la prossima istruzione da eseguire. Ad esempio:

```
1    // Valore: numero della riga contenente
2    // l'istruzione seguente
3    (LOOP)
4        istruzione 1...
5        istruzione 2...
6        istruzione 3...
```

Le seconde sono usate nelle A-instructions per identificare celle della memoria RAM a partire dalla 16-esima. Riceveranno quindi un valore che va dal 16 in poi. Ad esempio:

```
1    @counter        // Valore: 16
2    ...codice...
3    @max            // Valore: 17
4    ...codice...
5    @min            // Valore: 18
6    ...codice...
```

5 Procedura di assemblaggio

Il processo di assemblaggio è diviso in 3 fasi:

5.1 Inizializzazione

- si apre in lettura il file .asm in input;
- si inizializza la symbol table dei simboli predefiniti;

5.2 Prima passata

Si scorre l'input per inserire nella symbol table le codifiche delle etichette incontrate. Per farlo si usa un counter del totale delle A/C-instructions incontrate, assegnando a ogni etichetta il valore corrente del counter +1.

5.3 Seconda passata

Si apre in writing il file .hack prodotto e si scorre nuovamente l'input. Per ogni A/C-instruction incontrata ne si scrive nell'out la relativa codifica. Inoltre qualora una A-instruction usasse un simbolo, si dovrebbe ricercare questo nella symbol table e, nel caso in cui questo sia assente (quindi si sta parlando di una variabile) si assegna a questa istruzione un valore progressivo a partire da 16, per poi memorizzare questo nella symbol table.

5.4 Esempio di assemblaggio

N.B. questa non è una tipologia di esercizio ma bensì solamente un esempio per comprendere meglio quanto appena spiegato.

```
1     @x
2     M=1
3
4     (CICLO)
5         @1
6         D=M
7         x
8         D=D-M
9         @END
10        D;JLE
```

```
11          @CICLO
12          0;JMP
13
14      (END)
15          @END
16          0;JMP
```

Corrisponde alla tabella:

```
1       ...simboli predefiniti...
2       CICLO   2
3       END     15
4       x       16
```

Dove:

- CICLO ha valore di 2 in quanto prima della sua definizione si conta una sola A-instruction, quindi counter $+ 1 = 2$;
- END ha valore di 15 in quanto la riga seguente è la quindicesima;
- x ha valore di 16 in quanto è la prima (ed unica) variabile incontrata nel codice.