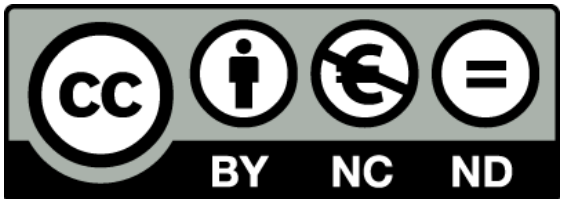
Laboratorio di Programmazione di Sistema

Dal Linguaggio C ai Linguaggi Assembly

Luca Forlizzi, Ph.D.

Versione 20.1



Luca Forlizzi, 2020

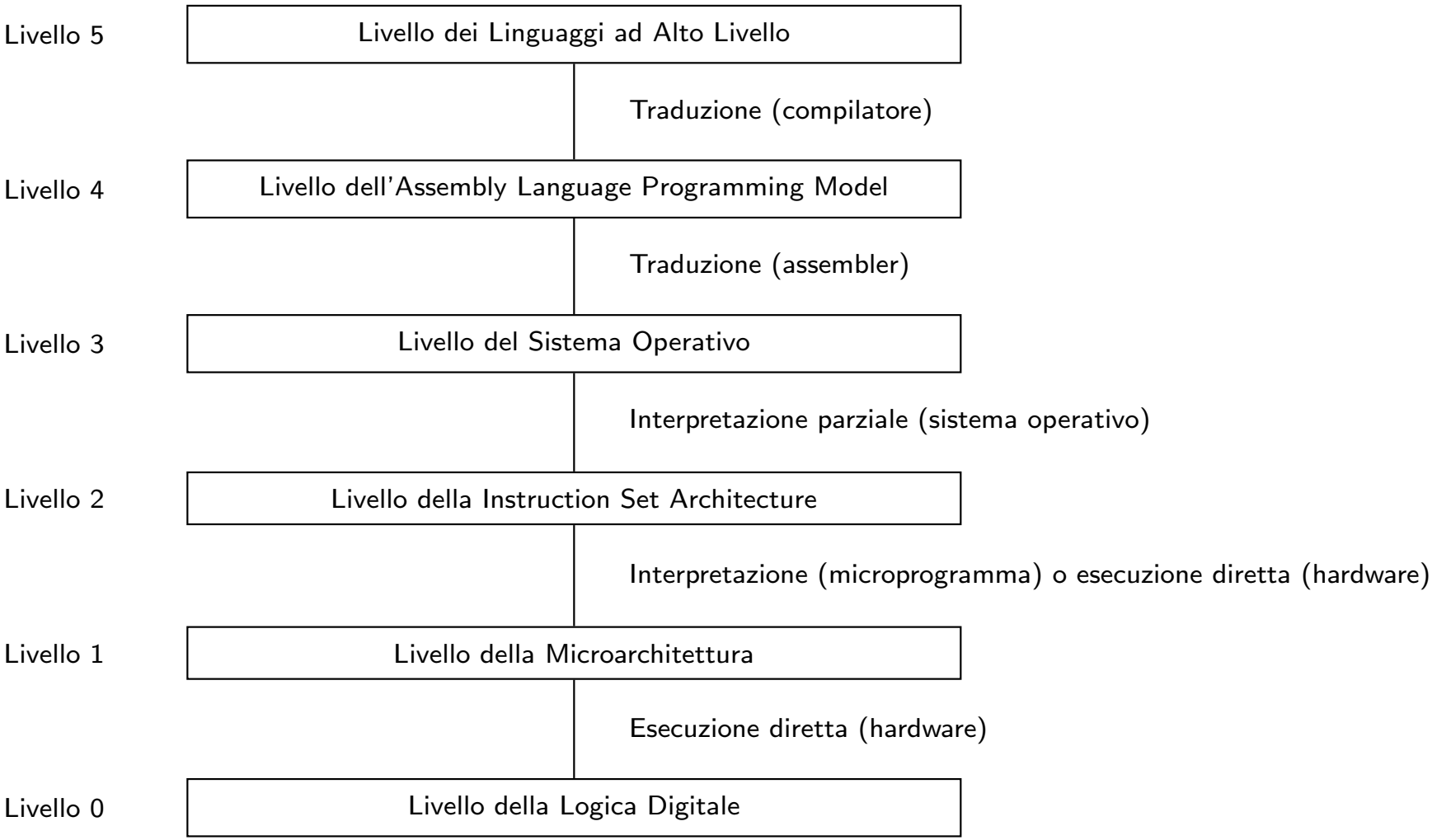
Semplificazione del modello a 6 livelli

- Nel modello concettuale adottato in LPS, un computer è un dispositivo di calcolo strutturato in 6 livelli di astrazione
- I livelli sono indicati con i numeri da 0 a 5
- Ad ogni livello troviamo una abstract machine M_i che esegue comandi di un linguaggio L_i, offrendo servizi ai livelli superiori

Semplificazione del modello a 6 livelli

- Una abstract machine di livello $i > 0$ viene implementata combinando abstract machine dei livelli sottostanti, ovvero utilizzando uno o più linguaggi L_h, dove $h < i$
- L'esecuzione di un programma scritto in un linguaggio L_i di livello i, con $i > 0$, può avvenire
  - traducendo il programma in uno equivalente scritto in un linguaggio L_h di livello $h < i$
  - oppure utilizzando un interprete scritto in un linguaggio L_h di livello $h < i$
  - se $h > 0$, per eseguire il programma scritto in L_h si ripete quanto fatto per quello scritto in L_i

Semplificazione del modello a 6 livelli



Semplificazione del modello a 6 livelli

- In LPS ci interessiamo di come un programma scritto in un linguaggio di livello 5 venga eseguito da una abstract machine di livello 2
- Concettualmente ciò avviene passando dal livello 5 al livello 4, dal 4 al 3 e dal 3 al 2
- In molte implementazioni del linguaggio C, il nostro linguaggio di livello 5 di riferimento, vengono realmente effettuati tutti questi passaggi

Semplificazione del modello a 6 livelli

- I linguaggi usati ai livelli 2, 3 e 4 di un computer sono molto legati tra loro, mentre quelli di livello 5 sono relativamente indipendenti da quelli dei livelli sottostanti
- Un linguaggio di livello 5 viene tradotto o interpretato in moltissimi linguaggi di livello 4 o inferiore
- Al contrario, un linguaggio di livello 4 viene di norma tradotto o interpretato in uno specifico linguaggio di livello 3
- A sua volta un linguaggio di livello 3 viene tradotto o interpretato in uno specifico linguaggio di livello 2

Semplificazione del modello a 6 livelli

- In ciascuno dei livelli 5, 4 e 3 possiamo distinguere dei *servizi di base* e dei *servizi avanzati*
- I servizi di base sono quelli fondamentali per la scrittura di un programma, quali
  - definire strutture dati elementari
  - effettuare calcoli elementari su dati interi o floating point
  - controllare la successione delle istruzioni eseguite
- Tra i servizi comunemente considerati avanzati troviamo
  - definire strutture dati sofisticate
  - effettuare operazioni I/O
  - effettuare calcoli complessi
- I servizi di base dei livelli 5, 4, 3 corrispondono al tipo di servizi offerti dal livello 2

Semplificazione del modello a 6 livelli

- A grandi linee troviamo una corrispondenza concettuale tra le componenti dei linguaggi ai diversi livelli

Livello	Servizi base	Servizi avanzati
5	Linguaggio C	C Standard Library
4	*ASM*	*ASM-API*
3	*LM**	*system call*
2	*LM*	

Semplificazione del modello a 6 livelli

- Individuiamo una corrispondenza analoga tra le componenti delle architetture ai diversi livelli

Livello	Servizi base	Servizi avanzati
5	C Standard freestanding	parti di C Standard hosted non contenute in C Standard freestanding
4	*ASM-PM*	*ASM-API architecture*
3	*ISA**	*operating system architecture*
2	*ISA*	

- In LPS concentriamo la nostra attenzione sulle componenti dei linguaggi che offrono i servizi base
- Le componenti che offrono servizi avanzati vengono studiate in *Sistemi Operativi con Laboratorio*

Semplificazione del modello a 6 livelli

- La corrispondenza concettuale tra i linguaggi si rispecchia in come viene effettuata la traduzione/interpretazione

Livelli	Costrutto	Traduzione/interpretazione
$5 \rightarrow 4$	costrutto C	programma *ASM*, di rado qualche comando *ASM-API*
$5 \rightarrow 4$	costrutto Standard Library	programma *ASM* e *ASM-API*
$4 \rightarrow 3$	singola istruzione *ASM*	poche (tipicamente da 1 a 3) istruzioni *LM**
$4 \rightarrow 3$	comando *ASM-API*	programma *LM** e *system call*
$3 \rightarrow 2$	singola istruzione *LM**	una istruzione *LM*
$3 \rightarrow 2$	singola *system call*	programma *LM*

Semplificazione del modello a 6 livelli

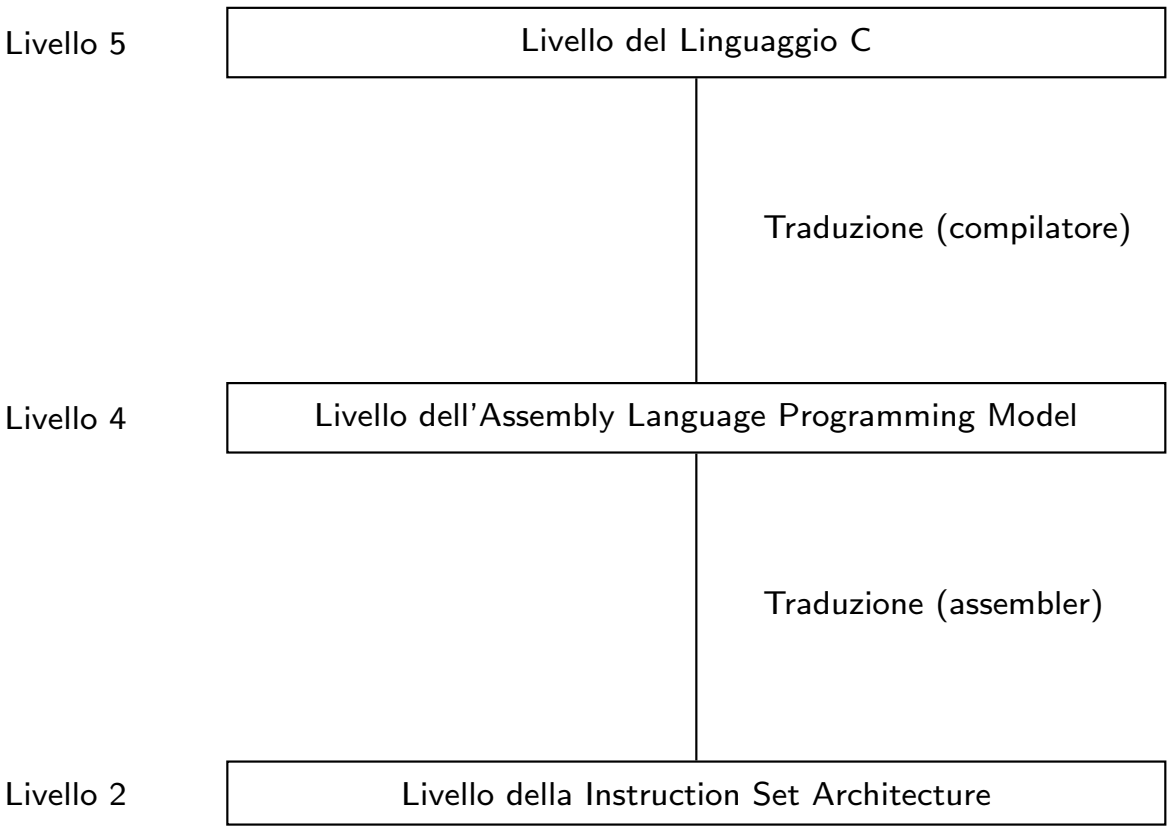
- Il livello 3, del sistema operativo, è un livello ibrido, in quanto i servizi di base che esso offre sono sostanzialmente quelli del livello 2
- In molti casi i servizi di base del livello 3 sono esattamente gli stessi del livello 2
- Relazione tra LM^* e LM
  - nella maggioranza dei casi, un'istruzione di LM^* è identica, sia nella sintassi che nella semantica, ad un'istruzione LM
  - in alcuni casi
    - alcune istruzioni di LM^* sono istruzioni di LM con leggere modifiche determinate dal sistema operativo
    - alcune istruzioni di LM non sono disponibili in LM^*
- Limitandoci a considerare la maggioranza dei casi, possiamo dire che il livello 3 "espone" il livello 2, rendendolo accessibile dai livelli superiori

Semplificazione del modello a 6 livelli

- Dunque, in relazione al livello 3, abbiamo la seguente situazione
  - in LPS non ci occupiamo di studiare i servizi avanzati del livello 3
  - LM^* (ovvero il linguaggio che fornisce i servizi base del livello 3) è uguale o molto simile a LM
- Pertanto nel resto del corso, salvo diversamente specificato, trascureremo il livello 3
- In altre parole assumeremo di utilizzare computer privi di sistema operativo
- Questa è una situazione che spesso si verifica in pratica nel mondo dei sistemi embedded
- In LPS, inoltre, non studiamo i livelli 0 e 1

Semplificazione del modello a 6 livelli

- Faremo quindi riferimento ad un modello di computer semplificato composto dai soli livelli 2, 4 e 5



Unstructured-C

- È talvolta utile scrivere programmi C secondo uno stile *non-strutturato* chiamato *Unstructured-C* (*UC*)
- Un programma *UC*, è un programma C che rispetta alcuni vincoli stabiliti per convenzione, allo scopo di rendere i programmi C più simili ai programmi *ASM* e *LM*
- Utilizzare *UC* è utile
  - come strumento per spiegare il funzionamento dei linguaggi *ASM*
  - come strumento operativo per tradurre un programma C in un programma *ASM* semanticamente equivalente
  - per illustrare alcuni aspetti del C stesso

Unstructured-C

- La trasformazione di un programma C in stile *UC* consiste nell'operare una serie di modifiche in modo tale da ottenere un programma che rispetti i vincoli dello stile *UC*, mantenendo la stessa semantica
- I vincoli definiti dallo stile *UC* hanno, in generale, lo scopo di de-strutturare i programmi in costrutti più semplici
- In questa lezione introduciamo i vincoli relativi alle dichiarazioni di variabile, all'istruzione `return` e alle *expression statement*

Unstructured-C

- Lo stile *UC* prevede per le dichiarazioni di variabile un solo vincolo: ciascuna variabile
  - non è inizializzata
  - oppure è inizializzata con un valore costante
- Lo stile *UC* prevede per l'istruzione `return` un solo vincolo: l'espressione che segue la keyword `return`
  - è vuota
  - oppure è una (singola) costante
  - oppure è una (singola) variabile

Unstructured-C

- Lo stile *UC* prevede per le *expression statement* solo le forme sintattiche presenti nella seguente tabella, in cui
  - `V` indica una variabile
  - `A` e `B` indicano una variabile oppure una costante
  - `F` indica una funzione
  - `L` indica una lista di parametri separati da virgole; ciascun parametro è una costante oppure una variabile oppure il simbolo `&` seguito da una variabile; la lista può essere vuota
  - op1 è uno degli operatori aritmetici unari `+`, `-`
  - op2 è uno degli operatori aritmetici binari `+`, `-`, `*`, `/`, `%`
  - op3 è uno degli operatori di assegnamento composto aritmetici `+=`, `-=`, `*=`, `/=`, `%=`

`V = A;`	`V = op1 A;`	`V = A op2 B;`
`V op3 A;`	`F( L );`	`V = F( L );`

Unstructured-C

- Per trasformare un programma C in stile *UC*, è quindi necessario destrutturare le espressioni aritmetiche e di assegnamento più complesse, senza alterare la semantica complessiva del programma
- A tale scopo può essere necessario introdurre delle variabili aggiuntive
- Ad esempio l'istruzione `a = x - y - a;` si può trasformare nel seguente codice *UC*

```
int t;
t = x - y;
t = t - a;
a = t;
```

Unstructured-C

- Si osservi che in *UC* non sono ammessi gli operatori `++` e `--`, né prefissi né postfissi
- Istruzioni formate solo dall'applicazione di uno di tali operatori ad una variabile si trasformano in modo molto semplice; ad esempio `x--;` si può trasformare in `x -= 1;`
- Istruzioni in cui gli operatori `++` e `--` compaiono all'interno di espressioni più complesse, richiedono più istruzioni
- Ad esempio l'istruzione `v = --x * y++;` si può trasformare nel seguente codice *UC*

```
x -= 1;
v = x * y;
y += 1;
```

Dal_C_agli_ASM: Indicazioni per Studio ed Esercizi

- Per studiare in dettaglio gli argomenti introdotti in questa presentazione
  - Capitolo 4 di **[Ki]** tranne alcuni contenuti che per il momento si consiglia di tralasciare (verranno presentati nelle prossime lezioni di LPS)
    - i concetti di *undefined behavior* e *implementation-defined behavior* che vengono introdotti nella sezione 4.1
    - i contenuti della sotto-sezione *Order of Subexpression Evaluation* che si trova nella sezione 4.4
- Svolgere gli esercizi collegati alla lezione **Dal_C_agli_ASM** su *Edu99*