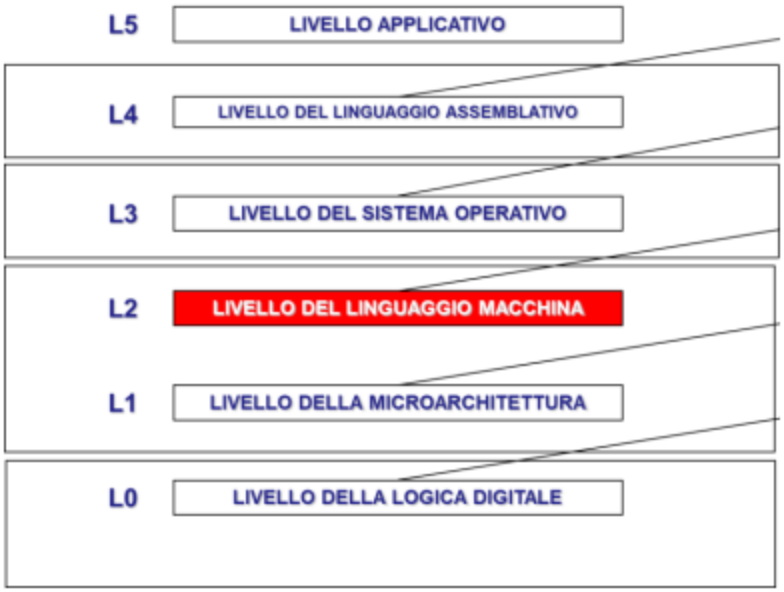
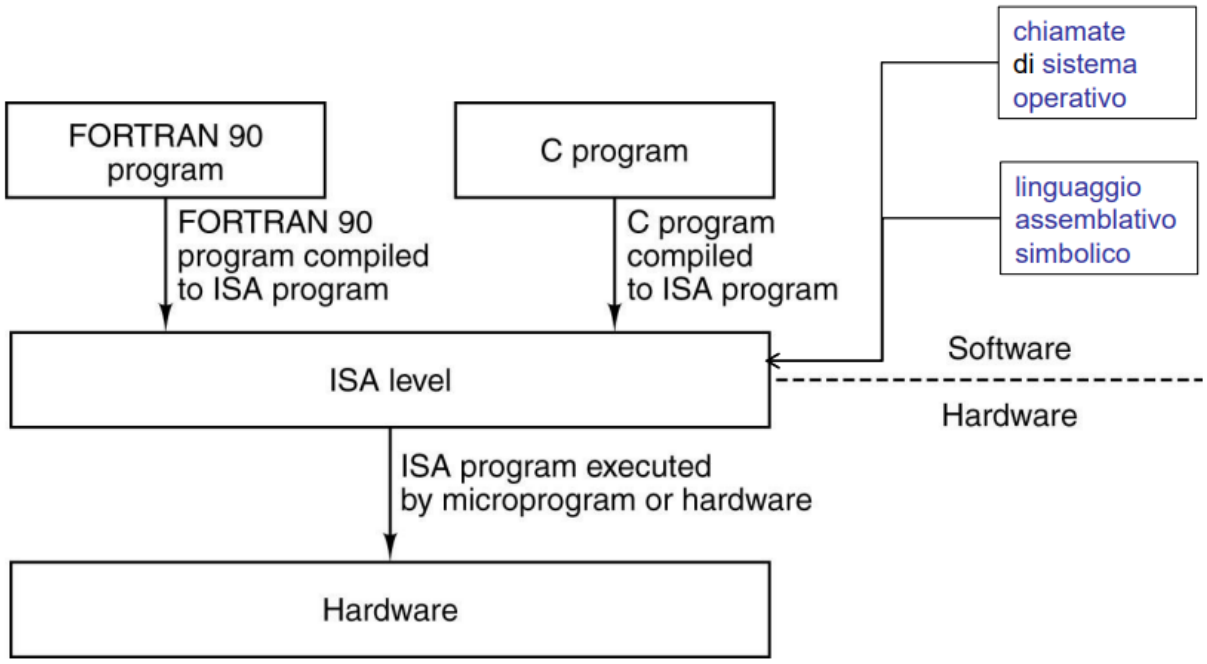
Linguaggio ISA

ISA è un acronimo per Instruction Set Architetcture.



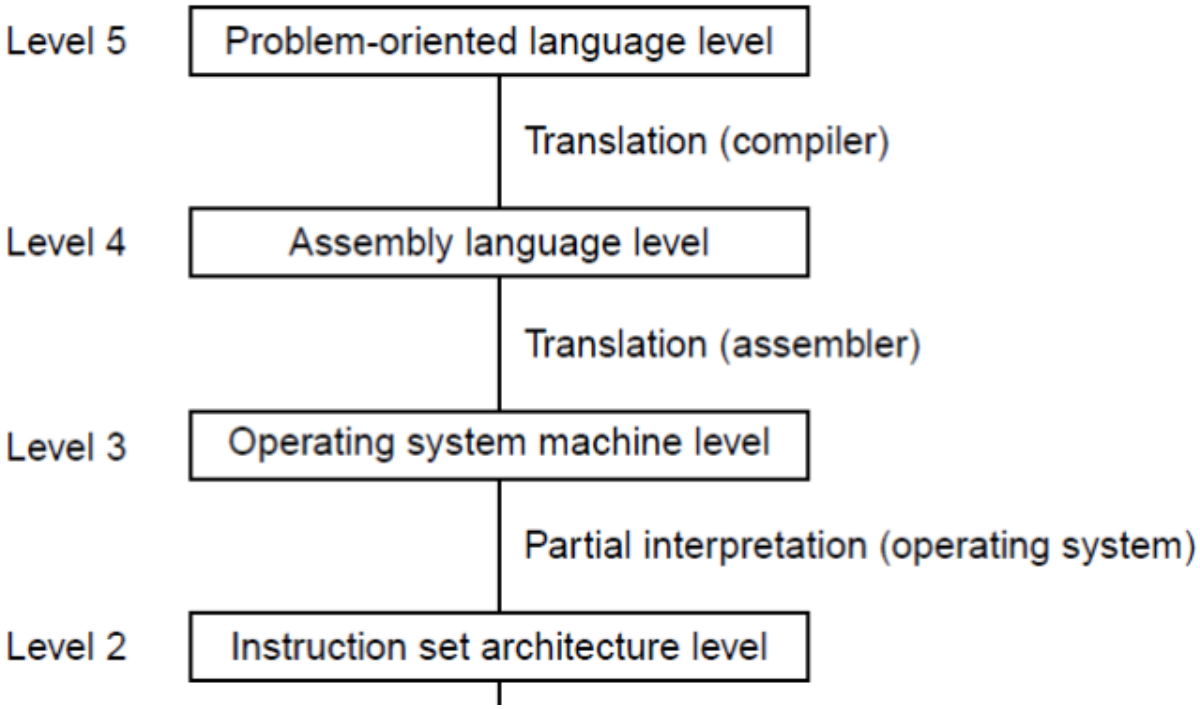
Cosa succede invece per la compilazione tra un linguaggio di livello più alto?



Idealmente il linguaggio macchina doveva essere un mezzo tra programmatori e progettisti. Nel mondo reale il linguaggio ISA viene adattato al **mercato**, si tende ad avere attenzione alla **backward compatibility**.

Il linguaggio ISA ha più livelli di utilizzo: **kernel mode** e **user mode.** La prima è una admin mode ovver il pieno controllo, la seconda ha molte limitazioni. Per alcune architetture il livello ISA è definito da un documento specifico, ha un vantaggio per i costruttori che realizzano la stessa architettura, è uno svantaggio per chi vuole mantenere una architettura proprietaria (esempio intal ha imnpiegato tempo prima di rilasciare la documentazione per ISA x86).

Il livello **ISA è quello che viene prodotto dagli assemblatori e dai compilatori** dei linguaggi ad alto livello (*nessuno programma più in linguaggio ISA*).



Il **linguaggio assemblativo** è una **versione simbolica di ISA** (ad esempio IJVM e MAL). Il linguaggio assemblativo ha accersso a tutte le istruzione della macchina: tutto ciò che può essere fatto in linguaggio macchina può essere fatto in liunguaggio assemblativo.

Ad esempio i programmi IJVM sono eseguiti dall'emulatore di MIC-1.

Anche ISA nasconde le micro-architettura, gli sviluppatori dei compilatori devono conoscere:

- Le caratteristiche di ISA:
  - com'è organizzata la memoria
  - quanti registri ci sono e a cosa sono dedicati
  - guali tipi di dati sono nativi sulla macchia
  - quali istruzioni sono disponibili
  - qual è il formato dekke istruzioni
  - quali sono le modiltà di indirizzmento
  - quali sono i maccanisni per la variazionie nel flusso di controllo

- Le caratteristiche hardware sottostante
    - pipeline
    - unità funzionali
    - organizzazione della cache

Com’è organizzata la memoria?

Le memoria è divisa in celle che hanno indirizzi consecutivi, l’orgnazzazione più comune prevede celle da 8 bit (8 bit è un byte, 4 byte o 8byte sono una word). Esistono istruzioni specializzate per manipolare byte e parole. Spesso si fa riferimento anche a gruppi di parole detti blocchi. In uesto caso non esistono dimensioni standerd ma è sempre convenutne che i blocchi siano formati da 2Kbyte.

Per migliorare la lettura è meglio che le parole siano indirizzate a multipli di 4 (terminano con 00) altrimenti parole di 8byte allineate su indirizzi multipli di 8 ovver che terminano con 000.

Un’altra caratteristica importante del livello ISA è la semintica delle operazioni di accesso alla memoria.

Emergono possibili comportamenti inattesi nel caso di architture in cui le istruzioni non sono eseguite in sequenza.

Ci sono per questo motivo tre possibili approcci per il problema →

- Lasciare al programmtore la responbilità di assicurare l’effettiva memorizzazione dei dati tramite una SYNC.
- Realizzare in hardware la verifica automatica della presenza di RAW (readAfter Write) o di WAR (Write After Read) nell’esecuzione di LOAD / STORE.
- Sequenzializzare tutte le LOAD / STORE.

Abbiamo lo spazio degli indirizzamenti ovvero l’insieme delle posizione delle istruzioni e dei dati di un programma definibili tramite le modalità di indirizzamento offerto da ISA.

La maggior parte delle archittture prevede un solo spaziodi indirizzamento lineare di byte, eventualmente raggruppati a parole. Indirizzi di k bit possono indrizizzare

da 0 a $2^k - 1$ bit.

Ci sono architetture che prevedono più spazi di indirizzamento lineari indipendenti (detti segmenti) per istruzioni e dati.

- Si può indirizzare spazi più grandi.
- Gli accessi possono essere più controllati spacificando i permessi per cianscun segmneto.
- I segmenti possono crescere o descrescere.

Ci sono due architetture di memorizzazione: **Big Endian** e **Little Endian**. Nel primo l'ndirizzo più basso è 00000000 e il più alto è 11111111. Viceversa per il Little Endian.

- Nel caso di una parola di 4 byte, il numero esadecimale 0x01234567 è memorizzato come:

Little Endian

0x...0011	...
0x...0100	0x67
0x...0101	0x45
0x...0110	0x23
0x...0111	0x01
0x...1000	...

byte meno significativo nella cella con indirizzo più basso

Big Endian

0x...0011	...
0x...0100	0x01
0x...0101	0x23
0x...0110	0x45
0x...0111	0x67
0x...1000	...

byte meno significativo nella cella con indirizzo più alto

Quanti registri ci sono e a cosa sono dedicati?

Tutti gli elaboratori dispongono di registri su cui opera la ALU. La loro funzione è quella di fornire una elevata accessibilità ai dati.

Normalmente ci sono poche decine di reguistri alcune dei queli spacilizzati ed altri a uso geerale. Normalmente i registri hanno dimensioni multipli di 8 bit.

Bisogna fare attenzione alla distinzione della visibilità tra i vari libelli: tutti visibilit al livello della microarchitettura alcuni visibili a livello ISA.

Un registro presente in tutte le architetture è il Flags Register o PSQ che contiene i condition code che sono asseriti dalla ALU:

- **N** - posto a 1 quando il risultato è negativo
- **Z** - posto a 1 quando il risultato
- **V** - posta a 1 quando causa overflow
- **C** - posto a 1 quando c'è riporto finale