Architettura degli elaboratori - lezione 8

Appunti di Davide Vella 2024/2025

Claudio Schifanella
claudio.schifanella@unito.it
Link al moodle :
https://informatica.i-learn.unito.it/course/view.php?id=3106

24/03/2025

Contenuti

Approfondimento con esempi sulle convenzioni nell'uso dei registri

Riassunto : non è una buona idea chiedere a ChatGPT.

Operandi immediati ampi

Problema : è possibile caricare in un registro una c ostante a 32 bit?

LUI

Load upper immediate, tipo U. Carica 20 bit più significativi della costante nei bit da 12 a 31 di un registro. Perché è importante? Perchè la semplice LI carica solo i primi 12 bit di un registro (valori da -2048 a +2047 infatti).
Altro problema, abbiamo messo i 20 bit più significativi sì, ma manca i primi 12, come si va? Basta usare un ORI (or immediate). I bit già impostati dalla LUI non vengono modificati e i primi 12 bit a 0 prendono il giusto valore.

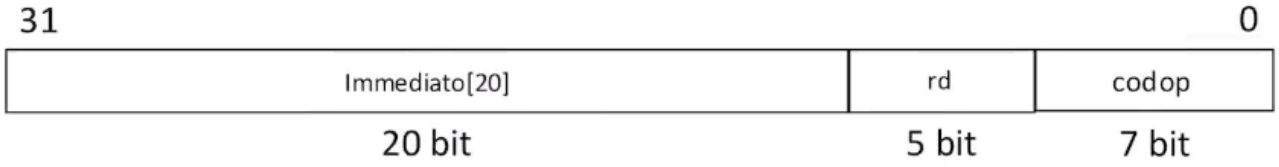
Salti con offset grandi

In RISC-V c'é la possibilità di saltare in un intervallo pari a 2^{32} grazie all'istruzione "auipc" (Add Upper Immediate PC) :

```
auipc rd, offset
```

Example

Formato U



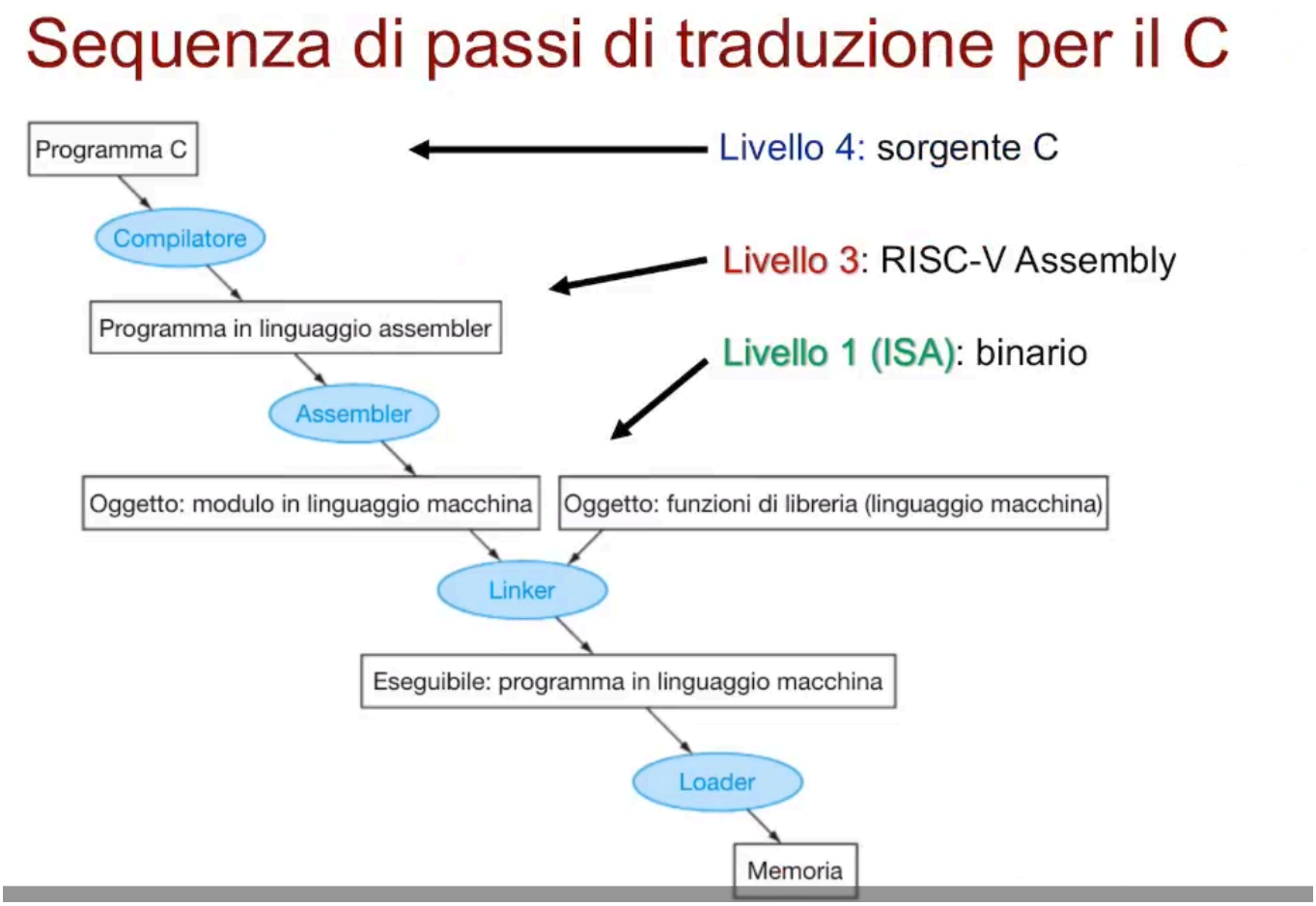
- immediato : 20 bit, quelli che vanno sommati al PC
- rd : register destination, da dove prendiamo i 20 bit
- codOp : codice univoco della operazione

Riassunto dei formati visti (importante per l'esame)

Nome (dimensione del campo)	Campi						Commenti
	7 bit	5 bit	5 bit	3 bit	5 bit	7 bit	
Tipo R	funz7	rs2	rs1	funz3	rd	codop	Istruzioni aritmetiche
Tipo I	Immediato[11:0]		rs1	funz3	rd	codop	Istruzioni di caricamento dalla memoria e aritmetica con costanti
Tipo S	immed[11:5]	rs2	rs1	funz3	immed[4:0]	codop	Istruzioni di trasferimento alla memoria (store)
Tipo SB	immed[12, 10:5]	rs2	rs1	funz3	immed[4:1,11]	codop	Istruzioni di salto condizionato
Tipo UJ	immediato[20, 10:1, 11, 19:12]				rd	codop	Istruzioni di salto incondizionato
Tipo U	immediato[31:12]				rd	codop	Formato caricamento stringhe di bit più significativi

> ⚠ **Warning**
>
> Non si devono ricordare a memoria tutte le posizioni di tutti i bit, ma è importante ricordare le considerazioni fatte. Ad esempio : "Perché il registro sorgente sta sempre nella stessa posizione? Per ragioni strutturali fisiche, è più facile costruire così", "Perché i campi dei registri sono grandi 5 bit? Ci sono 32 registri in tutto, $\log_2 32 = 5$, ovvero, ci bastano 5 bit per rappresentare tutti e 32 i registri in modo univoco", "perché codOp è sempre nella stessa posizione? CodOp, in modo particolare, è sempre nella stessa posizione perché è un campo in comune tra tutte le operazioni di tutte i formati, quindi è comodo averlo sempre nella stessa posizione in modo da sapere sempre dove si trova", "Perché l'immediato è grande 12 bit? Cosa significa avere 12 bit di immediato in un istruzione di salto? In una di tipo I?" ...

Sequenza di passi di traduzione per il C



Assemblatore (o Assembler)

L'assemblatore è un software che si occupa di trasformare il programma in assembly (a sua volta tradotto dal compilatore) in un modulo oggetto, cioè un file che contiene la codifica in linguaggio macchina delle istruzioni che abbiamo scritto in linguaggio mnemonico.

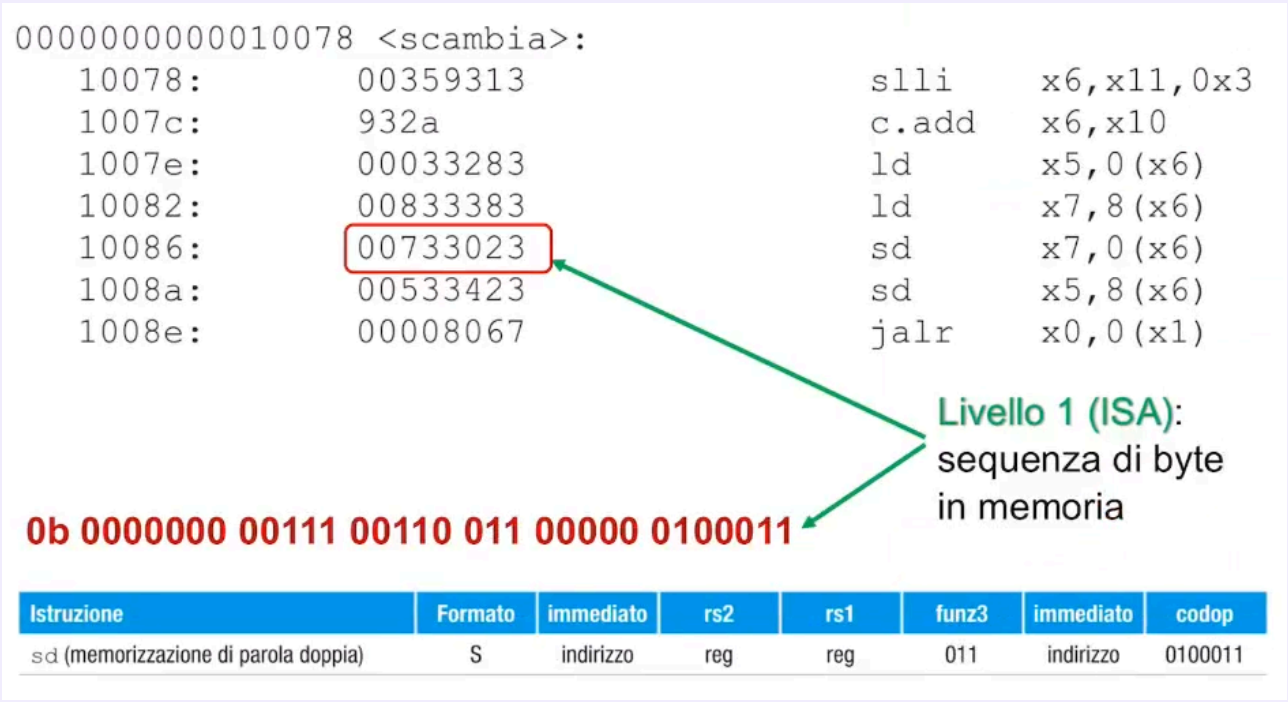
Linker

Software utilizzare dopo l'assemblatore per linkare i moduli in linguaggio macchina con le funzioni di librerie (linguaggio macchina) chiamate nel codice scritto. Un esempio può essere la printf() o la scanf(). Fatto ciò, il linker produce un eseguibile.

Loader

Software che si occupa di caricare l'eseguibile prodotto dal linker.

Example



Istruzione	Formato	immediato	rs2	rs1	funz3	immediato	codop
sd (memorizzazione di parola doppia)	S	indirizzo	reg	reg	011	indirizzo	0100011

Un esempio di come vengono tradotte le istruzioni (i codici sulla sinistra sono scritti in esadecimale)

Nota sul formato binario

Ovviamente, architetture diverse usano formati binari diversi, quindi incompatibili. Di seguito un esempio chiaro tra RISC-V e MIPS ISA :

Registro-registro

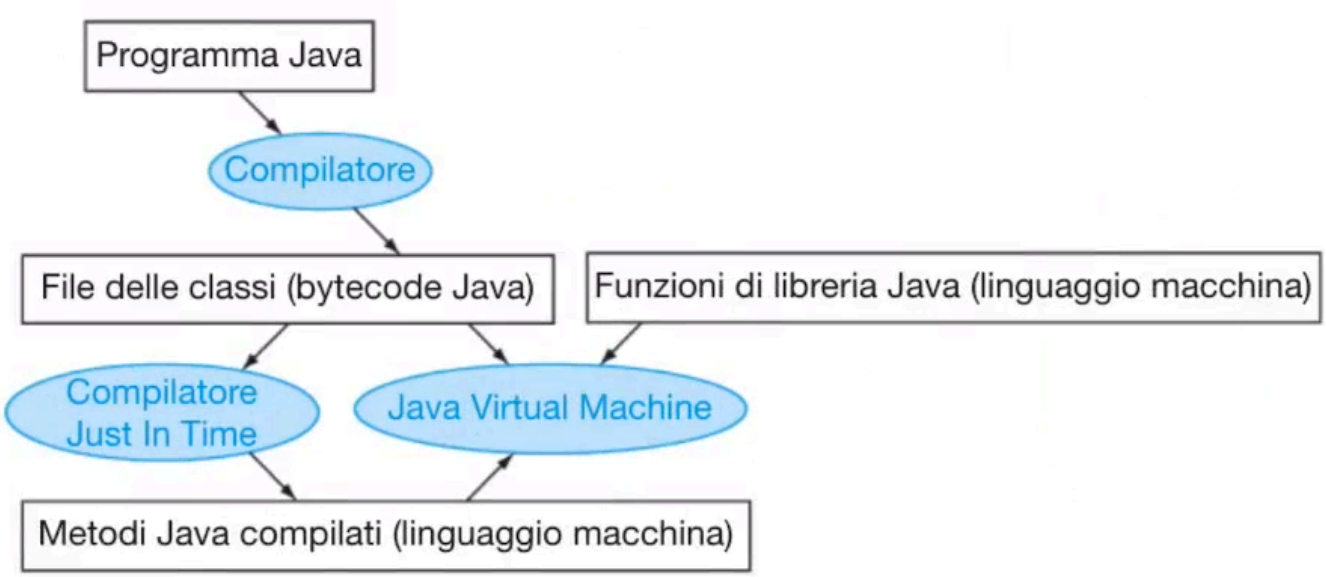
	31–25	24–20	19–15	14–12	11–7	6–0
RISC-V	funz7(7)	rs2(5)	rs1(5)	funz3(3)	rd(5)	codop(7)
	31–26	25–21	20–16	15–11	10–6	5–0
MIPS	Op(6)	Rs1(5)	Rs2(5)	Rd(5)	Cost(5)	Opx(6)

Trasferimento dalla memoria

	31–20	19–15	14–12	11–7	6–0
RISC-V	immed(12)	rs1(5)	funz3(3)	rd(5)	codop(7)
	31–26	25–21	20–16	15–0	
MIPS	Op(6)	Rs1(5)	Rs2(5)	Cost(16)	

Nota secondaria (non necessaria)

Linguaggi diversi hanno maniere diverse di venire tradotte, etc etc... Ad esempio, java non ha la stessa sequenza di passi di traduzione del C. Java usa una macchina virtuale per far girare il programma, quindi in Java avremo una struttura del genere :



Il programma in Java è quindi eseguito da un interprete (JVM - Java Virtual Machine), la quale può invocare il compilatore (JIT - Just In Time), il quale compila i metodi del linguaggio Java nel linguaggio macchina del calcolatore sul quale è in esecuzione. Ovviamente questo porta dei vantaggi sì, ma anche dei grandi svantaggi (ovvero l'inefficienza rispetto ad un programma scritto in C).

Linguaggio Assembly

Quando si parla di linguaggio Assembly si intende un linguaggio le cui istruzioni sono ottenute dalle istruzioni ISA sostituendo i codici binari con codici mnemonici; Il linguaggio Assembly è quindi molto vicino al linguaggio macchina. C'é sostanzialmente una corrispondenza uno-uno tra le istruzioni ISA e le istruzioni del linguaggio Assembly.
Ci sono delle facilitazioni nell'usare il linguaggio Assembly anziché scrivere in linguaggio macchina, ovvero :

- Si possono usare delle etichette simboliche per variabili e indirizzi
- La possibilità di usare primitive per allocazioni in memoria di variabili
- Costanti
- Si possono definire delle macro

> ⚠ **Warning**
>
> Attenzione, non ci si deve confondere :
>
> - Assembler : è il nome del software traduttore, NON è Assembly, sono due cose diverse.
> - Linguaggio macchina : NON è il linguaggio assembly o le istruzioni ISA, sono cose diverse.

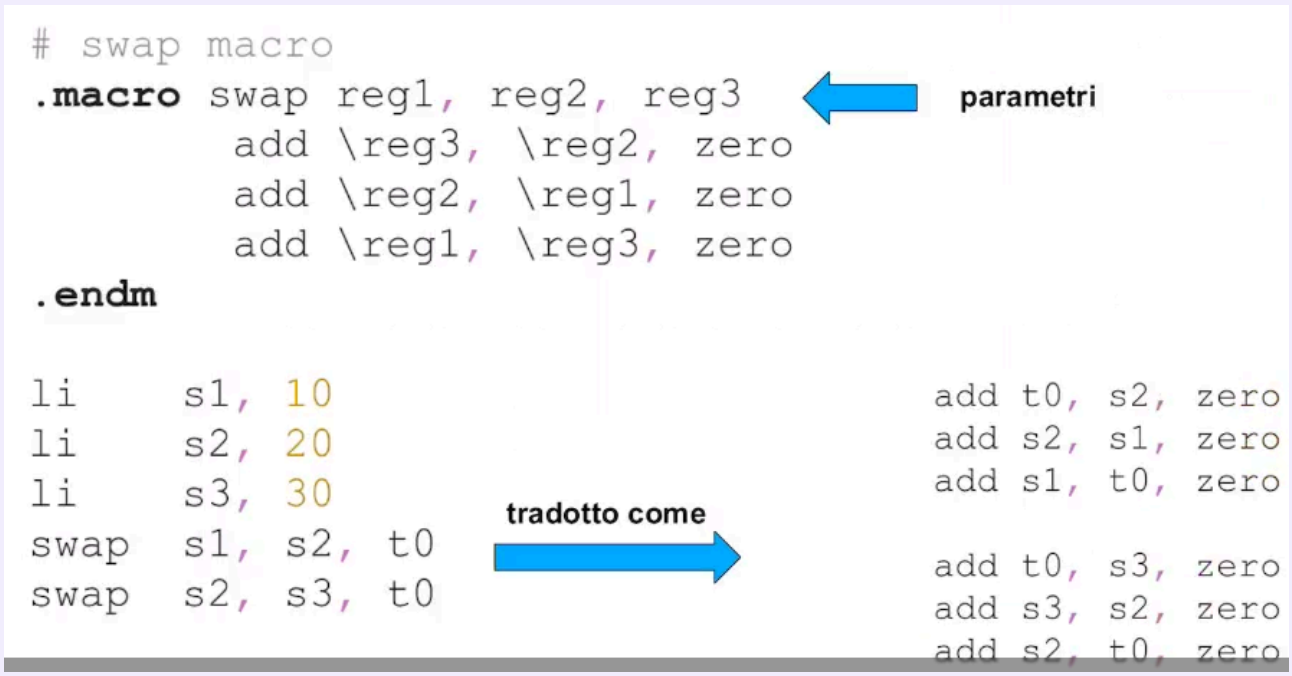
- Linguaggio Assembly : NON è uguale e univoco. È diverso per ogni architettura. Esiste quello Intel x86 (approfondimento disponibile sul libro di testo), esiste quello RISC-V (il quale ad esempio mette a disposizione il concetto di pseudo-istruzione, ovvero un'istruzione che in fase di traduzione, viene trasformata in più istruzioni, le direttive...)...

Macro (non viste in laboratorio, non essenziali in pratica, ma si devono sapere per teoria)

Una definizione di macro è un modo per assegnare un nome ad una sequenza di istruzioni. Dopo aver definito una macro il programmatore può scrivere il nome al posto della sequenza di istruzioni. Per definire una macro serve :

- un header della macro che indica il nome della macro da definire
- il testo che comprende il copro della macro
- una "assembly directive" che indica la fine della definizione

Example




```
# swap macro
.macro swap reg1, reg2, reg3
        add \reg3, \reg2, zero
        add \reg2, \reg1, zero
        add \reg1, \reg3, zero
.endm

li    s1, 10
li    s2, 20
li    s3, 30
swap  s1, s2, t0
swap  s2, s3, t0
```

```
add t0, s2, zero
add s2, s1, zero
add s1, t0, zero

add t0, s3, zero
add s3, s2, zero
add s2, t0, zero
```

Warning

Non stiamo parlando di procedure. Le macro non vengono tradotte in run time (ovvero quando eseguiamo il programma), ma è l'assemblatore che si occupa di tradurle.

Item	Macro call	Procedure call
When is the call made?	During assembly	During program execution
Is the body inserted into the object program every place the call is made?	Yes	No
Is a procedure call instruction inserted into the object program and later executed?	No	Yes
Must a return instruction be used after the call is done?	No	Yes
How many copies of the body appear in the object program?	One per macro call	One