Università di Firenze
Laurea Triennale in Informatica
Corso di Architettura degli Elaboratori
A.A. 2018/2019

Il Linguaggio Assembly del Sistema MIPS

Paolo Lollini
Dipartimento di Matematica e Informatica

lollini@unifi.it

Informazioni pratiche

- **Ripartizione delle ore**
  - ≈ 12 ore di lezione in aula
  - ≈ 10 ore di esercitazione (sempre in aula)

- **Esercitazioni**
  - Verrano svolti, con la supervisione del docente, **esercizi di programmazione assembly** col simulatore QtSpim;
    - QtSpim scaricabile al link: http://sourceforge.net/projects/spimsimulator/files/

- **Progetto di laboratorio**
  - *prerequisito* per svolgere l'esame finale (sia scritto che orale)

- **Riferimenti bibliografici**
  - D. A. Patterson, J. L. Hennessy. «Struttura e progetto dei calcolatori» – quarta edizione italiana condotta sulla quinta edizione americana
    - Capitolo 2 («Le istruzioni: il linguaggio dei calcolatori»)
    - Appendice A («Gli assemblatori, i linker e il simulatore SPIM»), che era Appendice B nella terza edizione italiana del libro)

Ore di esercitazione

- **Gli studenti svolgeranno le esercitazioni lavorando in gruppi di 2 o 3 persone**

- **Deve essere disponibile almeno un portatile per gruppo di lavoro, con batteria carica e con già istallato il simulatore MIPS (QtSpim o MARS).**

Modalità di consegna del progetto di laboratorio

- **Per sostenere l'esame è necessario consegnare preventivamente il codice e una relazione sugli esercizi assegnati**
  - Ogni gruppo di lavoro dovrà consegnare un'unica relazione
  - Il codice consegnato deve essere funzionante
- **Un archivio contenente il codice e la relazione dovrà essere caricato sul sito moodle del corso seguendo l'apposito link che verrà reso disponibile alla pagina del corso**
  - Non vengono prese in considerazione altre modalità di consegna
- **La scadenza esatta della consegna verrà resa nota di volta in volta**
- **Discussione e valutazione: la discussione degli elaborati avverrà contestualmente all'esame orale e prevede anche domande sull'assembly e su tutti gli argomenti di laboratorio trattati a lezione. La valutazione incide sulla votazione finale.**

Struttura della relazione (in formato pdf)

- **Info su autori e data di consegna**
  - gli autori (e loro indirizzo e-mail)
  - la data di consegna
- **Per ciascun esercizio**
  - Descrizione della soluzione adottata, trattando principalmente i seguenti punti:
    - Descrizione ad alto livello dell'algoritmo (in linguaggio naturale, con flow-chart, in pseudo-linguaggio, etc.), delle strutture dati utilizzate (liste, vettori, etc.) e delle procedure utilizzate (parametri, funzionalità svolte)
    - Uso dei registri e memoria (stack, piuttosto che memoria statica o dinamica)
    - Motivazione delle scelte implementative
  - Simulazione
    - In questa sezione andranno gli screenshot commentati di una o più simulazioni-tipo, anche discutendo l'evoluzione del contenuto del "user data segment" e dello "user stack" durante l'esecuzione del codice. Mostrare anche il funzionamento dell'algoritmo in corrispondenza di input sbagliati (es, inserisco da tastiera un carattere al posto di un numero).
  - Codice MIPS assembly implementato e commentato in modo chiaro ed esauriente.

Codice

- Il codice deve essere suddiviso in cartelle (una cartella per ogni esercizio) ed i nomi dei files devono permettere di identificare facilmente l'esercizio a cui si riferiscono. Unico file per esercizio.
- Ricordarsi di inserire anche nel codice gli autori (e loro indirizzo e-mail) e la data di consegna.
- Seguire fedelmente le convenzioni sull'uso della memoria e sulle procedure.
- Commentare il codice in modo significativo (`move $s4,$s3` non significa solo che "copio il contenuto del registro $s3 in $s4"....).

Suggerimenti...

- **Dato statistico: chi supera il progetto di AE alla prima consegna,**
  - ottiene (mediamente) le votazioni migliori all'esame di AE
  - ha la possibilità di sostenere tutti gli appelli di AE (fino agli appelli di febbraio 2019 compresi)
- **Quindi:**
  - Seguire attivamente le lezioni;
  - Andare periodicamente a ricevimento;
    - Sia per chiarire gli aspetti più «teorici»
    - Sia per chiarire gli aspetti più «implementativi», ad esempio per risolvere possibili problemi relativi agli esercizi di progetto
  - I ricevimenti sono su appuntamento, scrivere a *lollini@unifi.it*

Ulteriore aiuto attraverso i Tutor...

- I tutor attuali sono: *Irene Dini*, *Filippo Mameli*, *Chiara Rapicetta*, *Federico Schipani*
- Ricevono il lunedì ed il mercoledì dalle 14:00 alle 17:00
  - Stanza dottorandi al primo piano di Viale Morgagni 65 (dal citofono chiamare l'interno 1483)
- Per prendere un appuntamento usate il contatto email:
  - dimai-tutor-informatica-l@unifi.it

Argomenti

- **Linguaggi ad alto livello, linguaggio assembly e linguaggio macchina.**
- **Gerarchia di trasformazione**
- **MIPS: istruzioni aritmetiche, di trasferimento dati, di controllo e salto**
- **MIPS: modi di indirizzamento**
- **Gestione delle procedure e della memoria in MIPS. Uso della pila (stack)**
- **Sintassi di un programma in assembly MIPS. Esempi di programmazione**
- **Compilatore, Assembler, Linker e Loader**
- **Il simulatore QtSpim**

Challenge: quale è il significato di questa sequenza di bit ???

➔**00000010001100100100000000100000**

E se utilizzassi una rappresentazione simbolica di ***basso livello****?*

➔ **add $t0 $s1 $s2**

E se utilizzassi una rappresentazione simbolica di ***alto livello****?*

➔ **A=B+C**

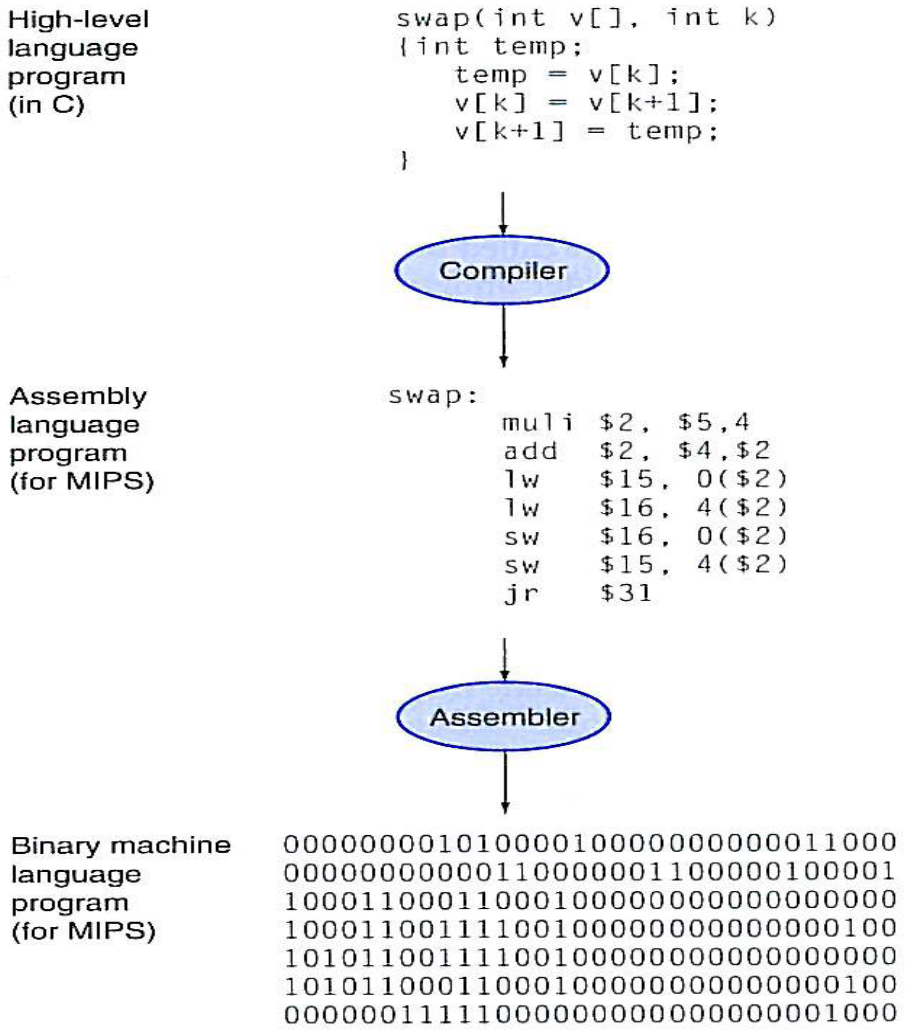
Introduzione

- **le CPU lavorano con linguaggio macchina**
  - sequenza di cifre binarie
- **Il linguaggio assembly è la rappresentazione simbolica (mnemonica) della codifica binaria usata dal calcolatore (linguaggio macchina)**
- ***simboli* al posto di sequenze di bit, fra cui:**
  - codici delle istruzioni (*opcodes*)
  - *registri*
- Esempio: `lw $t0, 100($t1)`

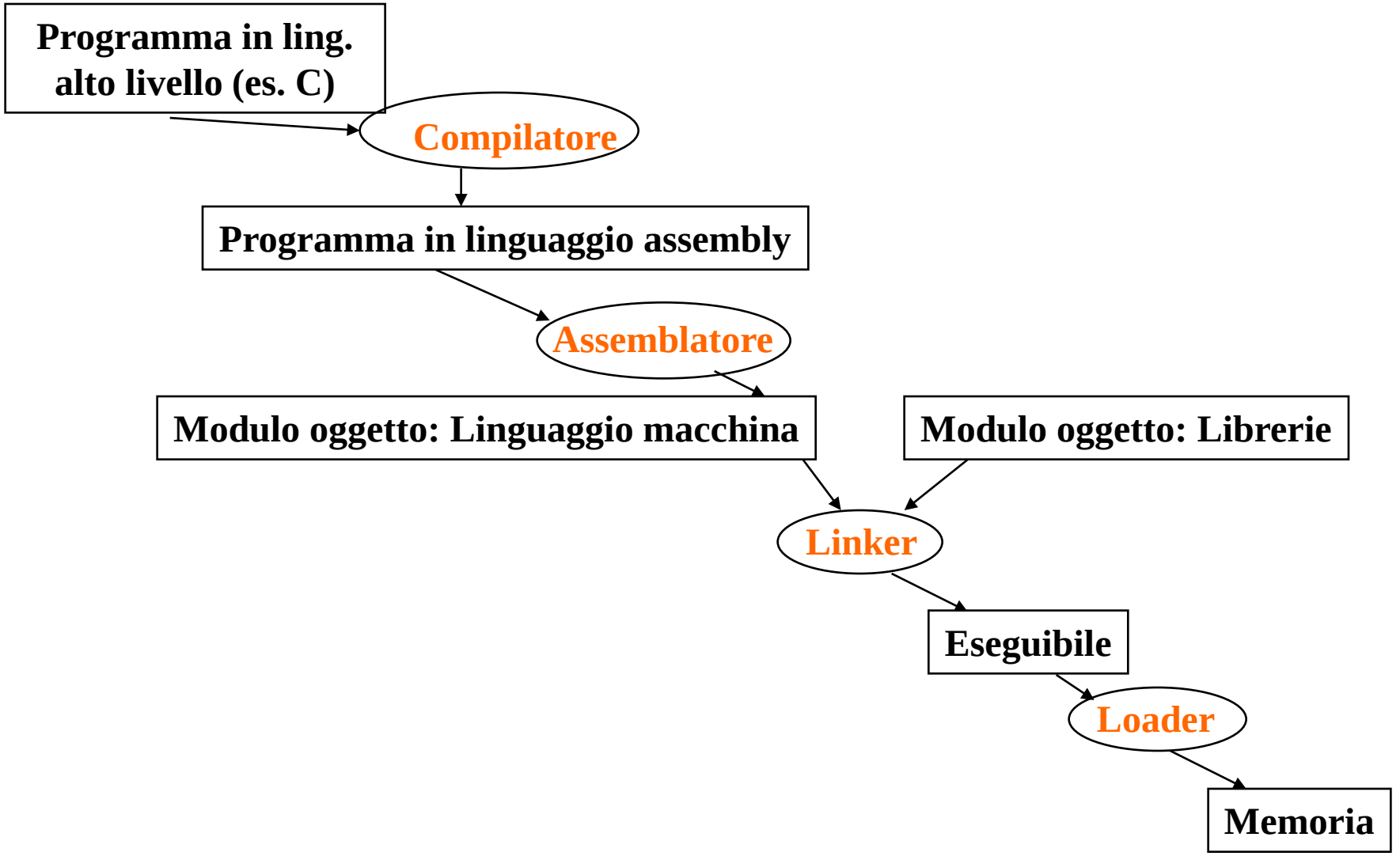
Traduzione dei programmi

- **Dal linguaggio ad alto livello tramite il compilatore il programma viene tradotto in linguaggio assembly**

- **E poi tramite l'assemblatore in codice macchina: il linguaggio direttamente comprensibile dal calcolatore.**



Gerarchia di trasformazione



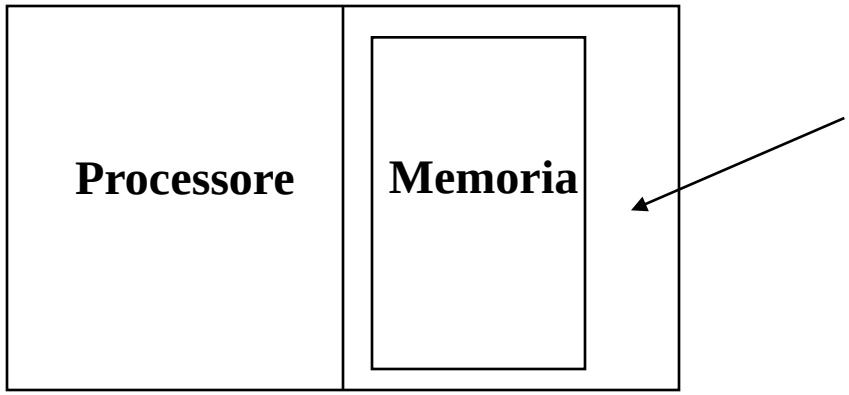
Gerarchia di trasformazione

Un programma scritto in linguaggio ad alto livello viene:

- **compilato in linguaggio assembly;**
- **assemblato per ottenere uno o più moduli oggetto in linguaggio macchina;**
- **il linker combina uno o più moduli oggetto tra loro e con le procedure contenute nelle librerie, risolvendo i riferimenti non definiti in ciascun modulo;**
- **il loader carica il linguaggio macchina in opportune locazioni di memoria, in modo che possa essere eseguito dal processore.**

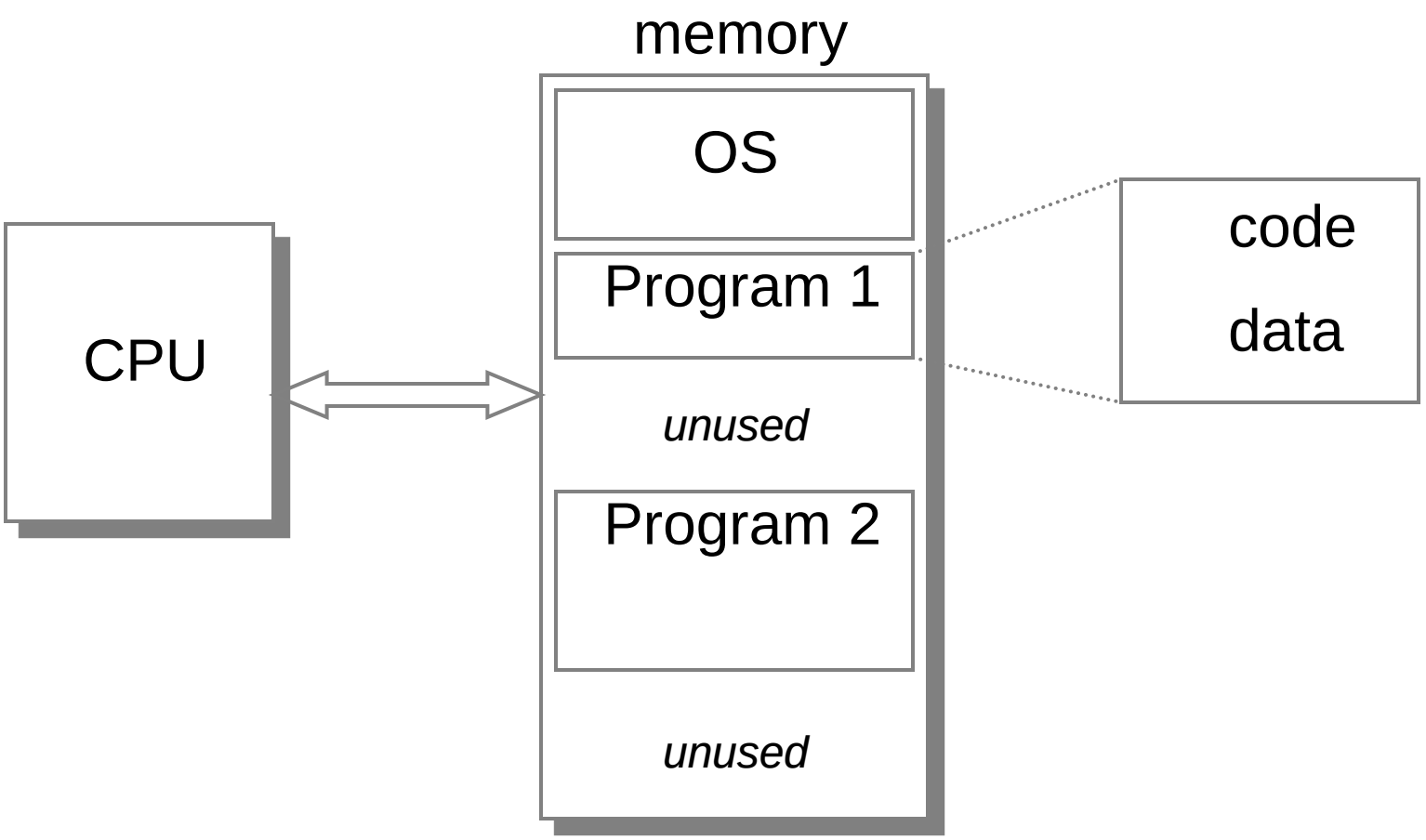
Concetto di programma memorizzato

- **Le istruzioni (e i dati) sono codificate come sequenze di bits**
- **I programmi sono memorizzati in memoria**
  - semplificazione hw/sw dei calcolatori



memoria per dati, programmi, compilatori, editor, etc

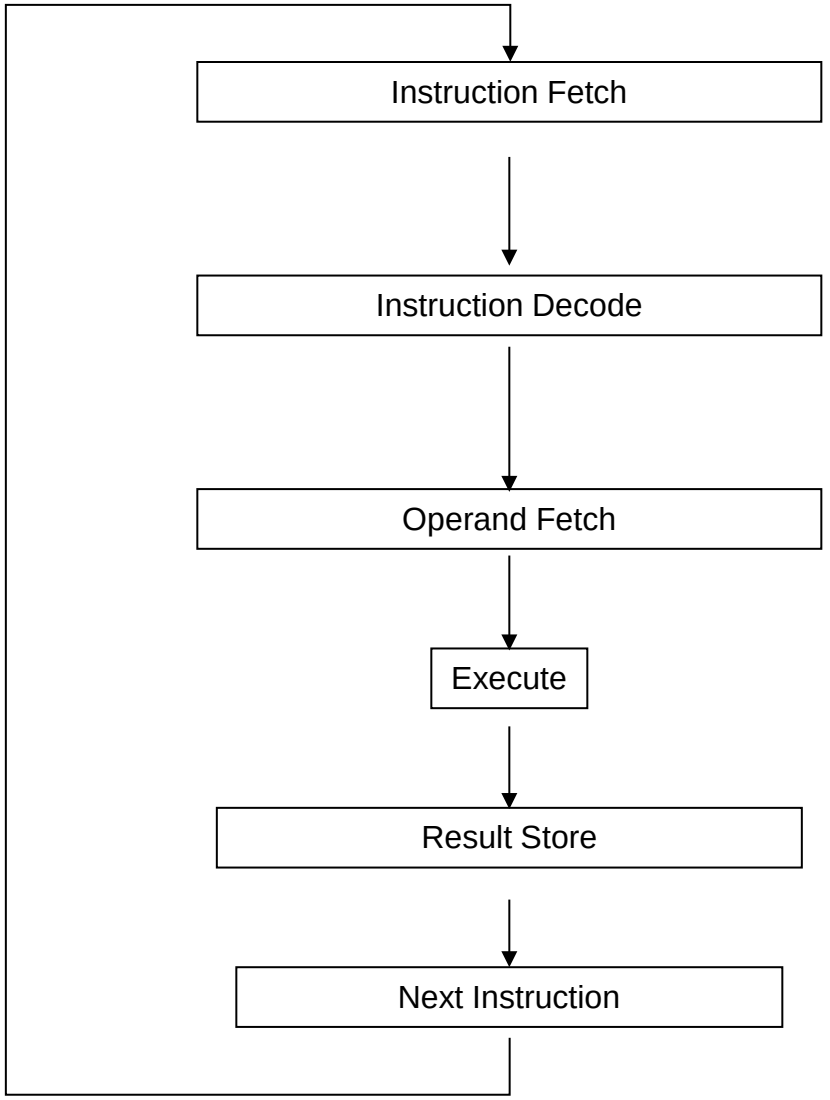
Programma memorizzato



Ciclo fetch, decode & execute

Ciclo Fetch, Decode & Execute

- l'istruzione corrente viene letta e messa in uno speciale registro interno
- i bit nel registro "controllano" le azioni seguenti
- viene letta la nuova istruzione e cosi' via



Linguaggio alto livello, assembly, macchina

Etichetta: **sequenza di caratteri alfanumerici minuscoli seguiti dal simbolo :(due punti). Rappresentazione simbolica degli indirizzi di istruzioni e dati.**

Direttiva: **istruzione che indica all'assemblatore come tradurre un programma.**

Programma che calcola e stampa la somma dei quadrati dei primi 100 interi

```
#include <stdio.h>
int main ()
{
int i;
int sum = 0;
for (i = 0; i <= 100; i = i + 1) sum = sum + i * i;
printf ("The sum from 0 .. 100 is %d\n", sum);
}
```

Differenze?

```
                .text
                .align 2
                .globl main
main:           subu $sp, $sp, 32
                sw $ra, 20($sp)
                sd $a0, 32($sp)
                sw $0, 24($sp)
                sw $0, 28($sp)
loop:           lw $t6, 28($sp)
                mul $t7, $t6, $t6
                lw $t8, 24($sp)
                addu $t9, $t8, $t7
                sw $t9, 24($sp)
                addu $t0, $t6, 1
                sw $t0, 28($sp)
                ble$t0, 100, loop
                la $a0, str
                lw $a1, 24($sp)
                jal printf
                move$v0, $0
                lw $ra, 20($sp)
                ddu $sp, $sp, 32
                jr $ra
                .data
                .align 0
str:
.asciiz "The sum from 0 .. 100 is %d\n"
```

```
00100111101111011111111111100000
10101111101111110000000000010100
10101111101001000000000000100000
10101111101001010000000000100100
10101111101000000000000000011000
10101111101000000000000000011100
10001111101011100000000000011100
10001111101110000000000000011000
00000001110011100000000000011001
00100101110010000000000000000001
00101001000000010000000001100101
10101111101010000000000000011100
00000000000000000111100000010010
00000011000011111100100000100001
00010100001000001111111111110111
10101111101110010000000000011000
00111100000001000001000000000000
10001111101001010000000000011000
00001100000100000000000011101100
00100100100001000000010000110000
10001111101111110000000000010100
00100111101111010000000000100000
00000011111000000000000000001000
00000000000000000001000000100001
```

Vantaggi e svantaggi del linguaggio assembly

- **Vantaggi**
  - velocità d'esecuzione (es.: calcolatori *embedded*)
  - occupazione di memoria
  - possibilità di usare istruzioni specializzate per sfruttare funzionalità hw che non hanno una corrispondenza nei linguaggi ad alto livello

- **Svantaggi**
  - no portabilità
  - rapporto di espansione elevato => minore produttività
  - minore leggibilità
  - no programmazione strutturata
  - dubugging difficoltoso

Linguaggio assembly

- **Più primitivo dei linguaggi di alto livello**

  - no controllo di flusso sofisticato
  - tipi di dati molto primitivi (es. interi $0..2^{32}-1$, floating point)
  - istruzioni con formato rigido (es., **add** ha *sempre* 3 operandi di tipo registro)

- **Noi lavoreremo con l'architettura del set di instruzioni MIPS**
  - RISC, simile ad altre architetture sviluppate a partire dagli anni '80
  - usata da NEC, Nintendo, Silicon Graphics, Sony
  - RISC: *Reduced Instruction Set Code*. Da contrapporre alla filosofia *CISC (Complex Instruction Set Code)*

- **Obiettivi di progetto di MIPS:** massimizzare la performance e minimizzare il costo, ridurre il tempo di progettazione

Linguaggio macchina

- **E' l'insieme delle istruzioni direttamente eseguibili dall'hardware. Ogni istruzione corrisponde ad una *operazione*.**
- **Ogni istruzione deve specificare che tipo di *operazione* è richiesta e quali sono i suoi *operandi***
  - no nomi simbolici, etichette, … : solo sequenze di istruzioni
  - istruzioni assembly codificate numericamente come **word** di **32 bit**; es:

    `add $t0 $s1 $s2` corrisponde a

    `00000010001100100100000000100000`
- **Una sequenza di istruzioni costituisce un programma**

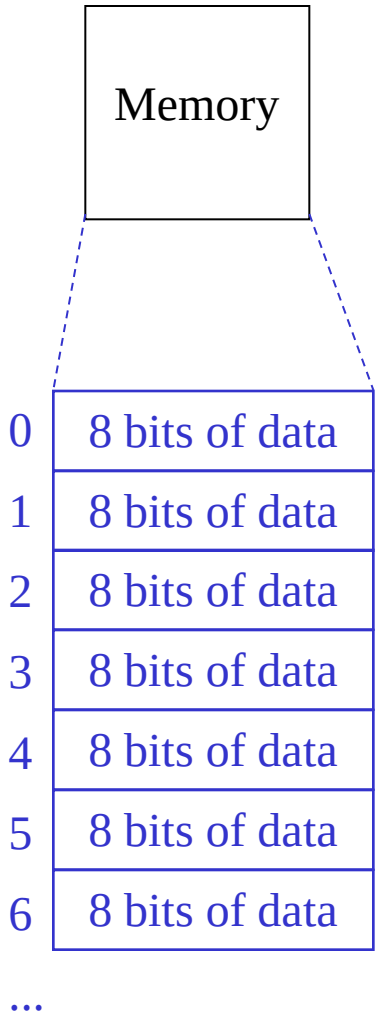
Registri e Memoria

- **Le istruzioni aritmetiche operano su registri di 32 bit**
  - solo 32 registri
  - ogni registro 1W (=4 byte)
- **il compilatore associa variabili a registri**
- **Cosa succede con programmi con tanti dati (variabili, array)?**
  - usiamo la memoria, che contiene anche i programmi

Organizzazione della memoria

- **La memoria MIPS è indirizzata al byte**

- **L'indirizzamento al byte è comodo, ma i dati sono memorizzati in "words" o "parole" (32 bits o 4 bytes)**
  - L'indirizzo di una parola corrisponde all'indirizzo di uno dei 4 bytes che la compongono
  - L'indirizzo di due parole consecutive differisce di 4 unità



Vincolo di allineamento

- **MIPS:**
  - **le parole iniziano sempre ad indirizzi multipli di 4 *(vincolo di allineamento)***
- **Organizzazione della memoria:**
  - 2^{32} bytes con indirizzi da 0 a 2^{32}-1
  - 2^{30} words con indirizzi 0, 4, 8, ... 2^{30}-4

0	**32 bits of data**
4	**32 bits of data**
8	**32 bits of data**
12	**32 bits of data**

Big Endian vs. Little Endian

- **Ordinamento dei byte in memoria**
  - **Big Endian** ***(MIPS, Sparc, Motorola 68k, ...)***
    byte più significativo ha indirizzo più basso
  - **Little Endian** ***(Intel 80x86, DEC Alpha, ...)***
    byte più significativo ha indirizzo più alto
- **Supponiamo di dover salvare in memoria la sequenza di bit $90AB12CD_{16}$ a partire dall'indirizzo 1000**

Big Endian

Indirizzo	Valore
1000	90
1001	AB
1002	12
1003	CD

Little Endian

Indirizzo	Valore
1000	CD
1001	12
1002	AB
1003	90

Registri

- **Gli operandi delle istruzioni sono contenuti in registri della CPU, *non* in celle di memoria**
  - tempo accesso ai registri è minore
  - throughput migliore

- **In memoria si memorizzano...**

le variabili meno usate

Convenzioni sull'uso dei registri

Registro: locazione di memoria elementare all'interno del processore

Nome	Numero	Uso
$zero	0	costante 0
$v0-$v1	2-3	calcolo di espressioni e risultati di una funzione
$a0-$a3	4-7	argomenti
$t0-$t7	8-15	temporanei non preservati dalle proc.
$s0-$s7	16-23	temporanei preservati dalle procedure
$t8-$t9	24-25	altri temporanei non preservati
$gp	28	global pointer
$sp	29	stack pointer
$fp	30	frame pointer
$ra	31	return address
$at	1	riservato all'assemblatore
$k0	26	riservato per il kernel del S.O.
$k1	27	riservato per il kernel del S.O.