Università degli Studi di Milano «La Statale»
Dipartimento di Informatica

Parte di pratica:

Esempi di programmi in MIPS
(su una macchina virtuale)

Marco Tarini

1

Problema pratico per questo corso

- Vogliamo provare i nostri programmi in MIPS
  - che scriveremo in assembly MIPS-32,
  - e che l'assembler traddurrà in linguaggio macchina MIPS-32
- Dove troviamo una macchina MIPS?
  - cioè, un HW in grado di eseguire un programma scritto in linguaggio macchina MIPS-32?

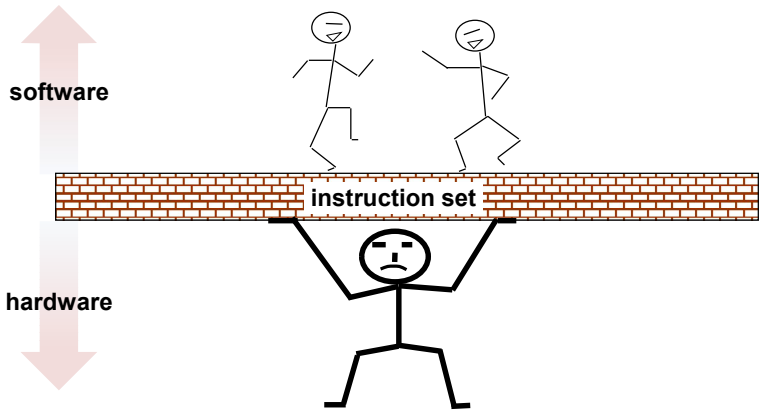
2

Emulazione
(= Instruction-Level Simulation)

- Ho una macchina HW per un Instruction Set *A,*
  ma i miei SW sono scritti per un diverso Instruction Set *B*
  - Software = programmi (binari, in linguaggio macchina B) + dati
- Soluzione: scrivo un interprete
  (un programma SW che gira su *A*)
  per un un IS *B*
  - Questo interprete è in grado di eseguire le istruzioni,
    quindi i programmi, scritti nel linguaggio macchina *B*
- Ora ho una «***macchina virtuale***» per B!
  - posso eseguire programmi per *B* pur non avendo una macchina (*fisica*) costruita per capire *B*
  - sto **emulando** l'HW dell'istruction set *A*

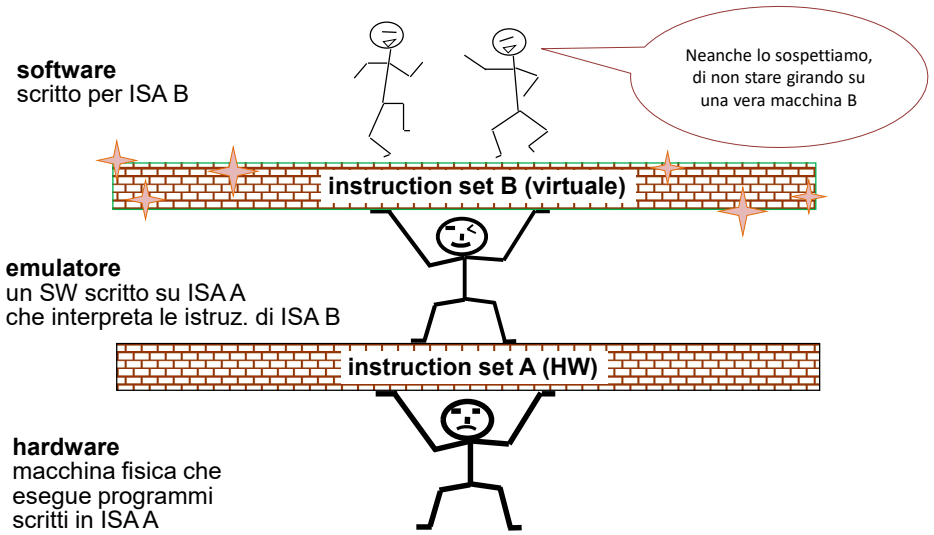
3

Il set di istruzioni: un'interfaccia critica



4

Macchina virtuale



5

Emulazione (= Instruction-Level Simulation)

Vantaggi:

- consente di riutilizzare il software
  (così com'è, alcun senza adattamento, o riscrittura)
  scritto per un'altra architettura B,
  senza avere l'HW che la esegue
- Chi scrive (o, ha scritto) il software per B non deve fare nulla di diverso dal solito

Svantaggi:

- La performance può risentirne:
  il programma emulato andrà molto più lento di quello originale che gira
  (emulazione software dell'esecuzione di una istruzione
  è molto più lenta della sua esecuzione su una CPU hardware!)

6

Macchine virtuali: le finalità

Il concetto di **macchina virtuale** è molto utile in diversi contesti:

- Rende il software binario **multipiattaforma** (**cross-platform**)
  - Grazie alle macchine virtuali, il codice scritto per un ISA può essere eseguito su macchine fisiche di un altro ISA
  - Vale anche retroattivamente, dopo la creazione del software (vedi ultimo punto)
- **Sicurezza** e **robustezza**
  - La macchina virtuale può introdurre controlli extra, ad es:
  - impedire l'accesso in lettura a file riservati
  - o proteggere i file di sistema dalla scrittura (accidentale o malevola).
  - In generale, la Virtual Machine è un "**sandbox**", un sistema chiuso che non può fare danni reali alla macchina reale (come cancellare tutto la mass storage del computer)
- **Testing** e **profiling**
  - La macchina virtuale può tener traccia di dati statistici su una data esecuzione di SW
  - Per es: quali comandi sono stati eseguiti più di frequente (e devono essere ottimizzati)? Con quali pattern è stata acceduta la memoria?
- **Conservazione** del software
  - Un software binario (in linguaggio macchina) scritto per un ISA obsoleto, supportato da un HW non più prodotto e non più esistente è perduto per sempre…
  - Una macchina virtuale riporta in vita il software del passato, emuando l'HW scomparso

7

Digressione: due macchine virtuali molto diffuse

Java Virtual Machine (**JVM**)

- Esegue «**bytecode Java**» che è …
- …un ISA (definizione completo di linguaggio macchina, RAM, banco di registri, etc) che non ha mai avuto macchine reali su cui girare (solo virtuali!)
- …il risultato della compilazione di un sorgente del linguaggio ad alto livello **Java**
- Provvisto di ottimizzazioni per migliorare le prestazioni (non le vedremo: si chiamano JIT - Just-In-Time)
- Provvisto di un ampio ecosistema (Java, Kotlin, Scala, ecc.)
- Suoi scopi principali: essere multipiattaforma, garanzie di sicurezza (è sandboxed)

JavaScript Engine (p.es. V8, SpiderMonkey)

- Una virtual machine integrata in tutti i web-browser
- Anche qui, esegue un «**bytecode**» (solo, non definito da uno standard, ma diverso per ogni implementazione)
- …che è il Risultato della compilazione di un sorgente del linguaggio ad alto livello **JavaScript** (che, occhio, non è parente di Java)
- Oltre alla Virtual Machine, l'engine include il complilatore/interprete di JavaScript
- Molte ottimizzazioni aggressive (JIT e altro)

8

Digressione: alcuni interessanti emulatori di ISA che puoi scaricare e provare

- **DosBox**:
  - emula l'IS X86, e il vecchio Sistema Operativo DOS, su varie piattaforme moderne (Windows, etc)
  - Consente di eseguire vecchi programmi DOS degli anni '80
- **WinE** (Windows Emulator):
  - emula varie piattaforme Windows per MacOS (emula il livello SO, non IS)
  - Consente di eseguire programmi Windows su Mac
- **MAME** (Multi Arcade Machine Emulator)
  - emula migliaia di IS di Arcade Machine (coin-op machine)
  - Consente di eseguire videogames arcade coin-op («da sala giochi») fine anni '70, anni '80, '90, 2000 e 2010
- **MESS** (Multi Emulator Super System) --- confluito in **MAME**
  - emula centinaia di IS di home gaming console (VIC-20, C-64, …)
  - consente di eseguire videogame scritti per gli ISA degli home entertainment systems.

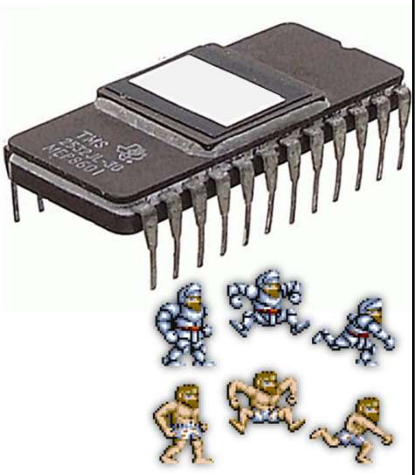
9

multi
arcade-machine
emulator

- Emulazione di migliaia di IS propri delle Architetture HW dedicate al gaming
  - coin-op dagli anni '70 ad oggi
- Software: i programmi originali per questi IS sono recuperati la' dove hanno aspettato per decenni: in chip di ROM
  - ROM dump = scaricare il contenuto di una (qui: vecchia) ROM
- Finalità: recuperare videogames, importanti pezzi della nostra storia culturale
  - è una corsa contro il tempo: ROM è memoria «persistente» sì… ma entro certi limiti temporanei!
  - quasi sempre, HW capace di eseguire quel dato IS non esiste più: Senza emulazione, molti video-games storici sarebbero perduti

10

multi
arcade-machine
emulator



- I programmi da eseguire in emulazione sono estratti da un insieme (10-50) di *chip di ROM*
  - capacità complessiva: molto bassa!
    Per es:
    pac-man (Midway, 1980): 20 KB
    ghost'n'goblin (Taito, 1985): 400 KB
- Come sempre:
  Programmi = DATI + ISTRUZIONI
- Nel caso dei videogame
  - ISTRUZIONI: il programma vero e proprio, scritto nell'IS della rispettiva architettura dedicate
  - DATI: includono i game "asset": sprites, effetti sonori, tilemaps, fonts, …

11

Macchine virtuali MIPS

Esistono alcuni progetti OpenSource:

- **MARS** (by Missuri State University)
- **Qt-SPIM**
- **MIPSWEB** (by UniMI)

Useremo quest'ultimo

- Seppure sia ancora WIP (Work In Progress)
- E' una web application!
- Qundi per usarlo basta accedere ad un URL:
  https://mipsweb.di.unimi.it/

12

Un primo programma MIPS

- Scriviamo un programma che calcola la somma di 2+3.
- Idea:
  - Inizializzare un registro qualsiasi con il valore 2
  - Inizializzare un altro registro qualsiasi con il valore 3
  - Effettuare la somma, ponendo il suo valore in un terzo registro qualsiasi
- Soluzione:

```
li $6 2
li $7 3
add $8 $6 $7
```

Da fare:

- Riporta la soluzione su MIPSWEB
- Assembla il programma (tasto in alto a destra)
- Osserva la traduzione delle pseudo istruzioni (nella RAM)
- Esegui, istruzione per istruzione (tasto Step in alto a destra)
- Osserva il risultato dell'esecuzione, guardando (nei registri)

20

Esercizio:

Scrivi, assembla, e testa (come sopra) un programma in assembly MIPS che

- Inizializza il registro $15 a 10
- Inizializza il registro $19 a 5.
- Implementa la soluzione dell'esercizio dell'ultima lezione, cioè calcola l'espressione:

$$\$20 = \left((\$15 + \$19) \ / \ (\$15 - \$19) \right)^2$$

21